

Java-natiivirajapinnan suorituskyky Androidin Dalvik-virtuaalikoneessa

Timo Tuominen

Helsinki 26.8.2016

Pro gradu -tutkielma

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen tiedekunta		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Timo Tuominen			
Työn nimi — Arbetets titel — Title			
Java-natiivirajapinnan suorituskyky Androidin Dalvik-virtuaalikoneessa			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level		Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Pro gradu -tutkielma		26.8.2016	83 sivua + 141 liitesivua
Tiivistelmä — Referat — Abstract			
<p>Android on mobiilikäyttöjärjestelmä ja sovelluskehitysalusta, jonka sovellukset ohjelmoidaan tavallisesti Java-kielillä ja suoritetaan Androidin omassa, Java-virtuaalikoneen kaltaisessa suoritusympäristössä. Tässä tutkielmassa käsitellään Androidin Dalvik-virtuaalikoneen toteutusta Javan natiivirajapinnasta (JNI), jonka avulla sovelluksissa voi yhdistellä Javalla sekä C- tai C++-kielellä toteutettuja komponentteja erityisesti suorituskykyvaatimusten saavuttamiseksi. Tutkielman tavoite on JNI-rajapintatoteutusta mittaamalla selvittää, voidaanko rajapinnan suorituskykykäyttäytymisestä laatia käyttökelpoinen malli, josta olisi hyötyä kun rajapintaa halutaan kutsua sovelluksesta optimaalisella tavalla. Tutkielma perustuu kahdenlaisiin suorituskykymittauksiin: rajapintaa harjoittavien operaatioiden vasteaikojen mittaamiseen sekä samojen operaatioiden kutsuprofilien laatimiseen ajonaikaisista kutsupinoista näytteenottotekniikalla. Mittaustuloksista johdetaan yksinkertainen lineaarinen malli, jossa on nähtävillä ne rajapinnan käytön osatekijät, joista rajapinnan suorituskykyrasitteet johtuvat. Mallin ja mittaustulosten perusteella voidaan nähdä, että virtuaalikoneen automaattisen muistinhallinnan asettamat vaatimukset aiheuttavat suurimmat suorituskykyrasitteet natiivirajapinnan käytössä. Ne johtuvat siitä, että muistinhallinnan on pidettävä kirjaa natiivikomponentille välitetyistä viitteistä tietorakenteissaan ja tarvittaessa estää muistialueiden siirtäminen muistiosoitteesta toiseen. Lopputuloksena todetaan, että mittauksia tulkitsemalla saatiin aikaan hyödyllinen suorituskykymalli, jota voidaan hyödyntää sovelluskehityksen varhaisessa vaiheessa, kun natiivirajapinnan käyttötapoja suunnitellaan.</p> <p>ACM Computing Classification System (CCS):</p> <ul style="list-style-type: none"> • Software and its engineering~Software performance • Software and its engineering~Runtime environments • <i>Information systems~Mobile information processing systems</i> 			
Avainsanat — Nyckelord — Keywords			
Suorituskyky, ohjelmistotekniikka, mobiilisovellukset, Android, Dalvik, Java, Java Native Interface			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

Sisältö

1	Johdanto	1
2	Yleiskatsaus Android-järjestelmään	4
2.1	Androidin rakenne	4
2.2	Dalvik-virtuaalikone	6
2.3	ART-virtuaalikone	8
3	Suorituskyky ohjelmistokehityksessä	9
3.1	Suorituskyvyn hallinta tuotantoprosessissa	9
3.2	Suorituskykymittarit	11
3.3	Mittausmenetelmät	12
3.4	Mittauskohteen edustavuus ja tulosten yleistettävyys	18
4	Java-natiivirajapinta	24
4.1	Java Native Interface ja vieraskutsurajapinnat	24
4.2	Natiivirajapinnan suunnitteluperiaatteet	25
4.3	Natiivirajapinta sovellusohjelmoijan näkökulmasta	26
4.4	Muistinhallinta natiiviohjelmoinnissa	32
5	Suorituskykymittaukset	35
5.1	Koeasetelman kuvaus	35
5.2	Mittautulokset	46
5.3	Dalvikin JNI-rajapinnan suorituskykymalli	72
6	Yhteenveto	75
	Lähteet	78
	Liitteet	
1	Mittausten generointi ja käsittely	
2	Muutokset Android-alustaan	
3	Mittausohjelmiston lähdekoodi	

1 Johdanto

Android on Googlen kehittämä Linux-pohjainen käyttöjärjestelmä ja sovelluskehitysalusta mobiililaitteille. Tässä tutkielmassa kuvaillaan ja arvioidaan sellaisten Android-sovellusten suorituskykyä, joiden komponenteista osa on kirjoitettu Java- ja osa C- tai C++-ohjelmointikielellä.

Yleensä Android-sovellukset ohjelmoidaan *Java*-ohjelmointikielellä ja suoritetaan Googlen Androidia varten kehittämässä virtuaalikoneessa. Androidin Java-sovelluskehityksen käyttäminen ei aina kuitenkaan ole kokonaisuuden kannalta paras ratkaisu. Ensinnäkin käytettävissä saattaa olla muulla kielellä kuin Javalla kirjoitettua ohjelmakoodia, jota halutaan hyödyntää sellaisenaan. Toiseksi täysin uudestakin mobiilisovelluksesta halutaan usein lähtökohtaisesti rakentaa mahdollisimman siirrettävä.¹ Kolmas ja tämän tutkielman kannalta oleellisin syy arvioida vaihtoehtoja Java-kielen käytölle on suorituskyky erityisesti, jos ohjelmisto suorittaa paljon raskasta laskentaa tai sille on oleellisia tosiaikavaatimuksia.

Ohjelmistojen suorituskykytekniikka (*Software Performance Engineering, SPE*) on ohjelmistotekniikan osa-alue, jossa varmistetaan ohjelmistoprosessin alusta lähtien, että ohjelmistolle asetetut suorituskykyvaatimukset täyttyvät lopullisessa tuotteessa [Smi90]. Menettely on näennäisesti ristiriidassa usein lainatun Donald Knuthin lausahduksen kanssa: "Ennen aikainen optimointi on kaiken pahan alku ja juuri [Knu74, s. 268]." Yleisen tulkinnan mukaan ohjelmistoa suunniteltaessa tulisi keskittyä sen oikeaan toimivuuteen ja optimoida suorituskykyä vasta, kun käytössä on suoritettava ohjelma. Sen suoritusaikaista käyttäytymistä mittaamalla selvitetään, mihin ohjelman osiin optimoinnissa kannattaa keskittyä.

Kehitysprosessin loppuvaiheessa ei kuitenkaan välttämättä enää ehdi korjata suorituskykyongelmia, jos ne kumpuavat varhaisista valinnoista kuten käytetyistä ohjelmointikielistä, ohjelmiston rakenteellisista ominaisuuksista ja komponenttien väliseen kommunikaatioon liittyvistä ratkaisuista. SPE:n keskeisiä väitteitä onkin, että ohjelmiston makrotason arkkitehtuurin suunnittelu on usein suorituskyvyn kannalta vähintään yhtä oleellinen tekijä kuin mikrotason optimoinnit [Smi90, s. 7], [WS98,

¹Esimerkiksi Applen *iOS*-käyttöjärjestelmän sovellukset käännetään *Objective-C*- tai *Swift* -kielestä laitteiston konekielellä suoritettaviksi, ja alustan kehitystyökaluilla voi suoraan hyödyntää myös C- tai C++-kielisiä komponentteja [App16]. Javan lupaus siirrettävyydestä ei realisoidu mobiilikäytössä, koska valmistajien kontrolloimiin järjestelmiin ei voi asentaa vaihtoehtoisia suoritussympäristöjä.

s. 164]. Suorituskyvyn huomioiminen varhaisessa kehitysvaiheessa ei toki estä lisä-optimointoja myöhemmässä kehitysvaiheessa.

Tutkielmassa keskitytään siihen, minkälaisia ylimääräisiä suorituskykyrasitteita *monikielinen* Android-ohjelmointi aiheuttaa sovellukselle, ja voidaanko näitä rasitteita mallintaa tavalla, josta olisi hyötyä jo varhaisessa sovelluksen kehitysvaiheessa.

Java-kielen rinnalla Androidia on mahdollista ohjelmoida myös käyttäen kohdealustana laitteiston omaa natiivisuoritusympäristöä. Tällöin ohjelmointi tapahtuu käytännössä C- tai C++-kielillä, joista ohjelma käännetään laitteiston prosessorin konekielelle, yleensä ARM-konekielelle. Menetelmän mahdollistaa Java-kielen *Java Native Interface* -standardi (*JNI*), joka on toteutettu myös osana Androidin Java-virtuaalikonetta. JNI-standardi on esimerkki *vieraskutsurajapinnasta* (*Foreign Function Interface, FFI*), joka on mekanismi kahden eri ohjelmointikielillä kirjoitetun ohjelman yhdistämiseksi.

Pelkän JNI-toteutuksen lisäksi Android-kehittäjällä on käytettävissään *Native Development Kit (NDK)*-työkalukokonaisuus. Sen rakennusskriptit huolehtivat C- tai C++-ohjelman kääntämisestä laitteiston kohdearkkitehtuurille sopivaksi kirjastobiinääriksi. Lisäksi NDK sisältää joukon C-kielisiä otsaketiedostoja ja ohjelmointirajapintoja, joiden kautta osajoukkoa Android-alustan ominaisuuksista voidaan hyödyntää suoraan natiivista ohjelmasta kutsumatta Java-sovelluskehityksen rajapintoja JNI-kerroksen kautta. Natiivitason Android-rajapinnoista ainoastaan nämä pysyvät vakaina alustan kehittyessä. Uusissa NDK-versioissa voidaan jopa ohjelmoida näitä rajoitettuja rajapintoja käyttävä sovellus kokonaisuudessaan ilman Javaa.

Suorituskyvyn kannalta Androidin natiiviohjelmoinnissa keskeistä on Java- ja natiivikomponenttien välinen kommunikaatio ja sitä kautta myös sovelluksen kokonaisarkkitehtuuri. JNI-kutsukerroksen käyttäminen on aina suorituskykyrasite ja erityisesti silloin, kun natiivikomponentti kutsuu Java-komponenttia. Monia haasteita liittyy myös muistinhallintaan ja resurssien jakamiseen virtuaalikoneen hallitseman muistin ja natiivikomponentin välillä.

Tutkielmassa käytetään suorituskykymittauksia tutkimaan monikielisydestä johtuvien rasitteiden suuruusluokkia. Mittaukset perustuvat Androidin versioon 4.1.2. Tutkielman aiheen kannalta se tarkoittaa erityisesti, että tutkielman mittaukset ja niiden tulokset koskevat Androidin edellisen sukupolven Dalvik-virtuaalikonetta.

Tutkielma sisältää kirjallisuuteen perustuvan osuuden, jossa esitellään suorituskyvyn mallintamiseen, arvioimiseen ja mittaamiseen käytettäviä periaatteita, metodei-

ta sekä työkaluja. Lisäksi lähteiden perusteella esitellään Android-ohjelmistoalustan rakennetta suorituskyvyn näkökulmasta keskittyen Dalvik-virtuaalikoneeseen ja sen JNI-toteutukseen. Tutkielman toinen osa soveltaa ensimmäisen osan metodeja ja käsitteitä Dalvikin JNI-toteutuksen mittaamiseen. Toteutusta mitataan kahdella toisinaan tukevalla menetelmällä, joiden tuloksista koostetaan yksinkertainen malli JNI-rajapinnan aiheuttaman ylimääräisen vasteajan keskeisimmistä osatekijöistä. Mallin avulla voi saada karkean suuruusluokka-arvion rajapinnan käyttämisen kustannuksista erilaisissa tilanteissa.

Tutkielman tavoite on saada selville, voiko Androidin virtuaalikoneen natiivirajapinnan toteutuksen suoritusajasta käyttäytymistä mittaamalla löytää riippuvuuksia ja lainalaisuuksia, joista olisi hyötyä varhaisessa kehitysvaiheessa olevan monikielisen sovelluksen suunnittelussa sellaiseksi, että se käyttäisi natiivirajapintaa suorituskyvyn kannalta optimaalisella tavalla.

2 Yleiskatsaus Android-järjestelmään

Tässä luvussa esitetään lyhyt yleiskuvaus tutkielman mittauskohteesta eli Android-järjestelmästä taustaksi myöhemmille luvuille, joissa syvennyttään tutkielman varsinaiseen aiheeseen, suorituskyvyn mittaamiseen ja mallintamiseen sekä Androidin Java-natiivirajapintaan.

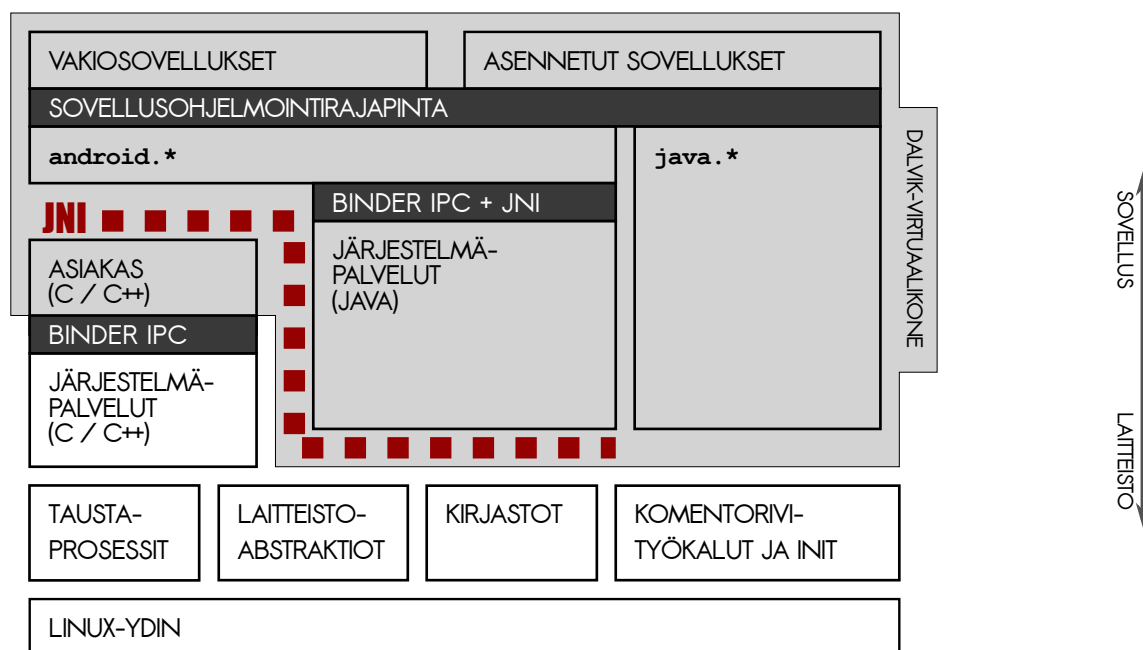
Androidin versiosta 6 lähtien Dalvik-virtuaalikone on korvattu ART-virtuaalikoneella. Koska tämän tutkielman mittaukset on tehty Dalvikia vasten, uudistettu virtuaalikone tekee mittaustuloksista osin vanhentuneita. Tutkielman metodologian kannalta uudistuksella ei kuitenkaan ole väliä. Samat mittaukset voi toteuttaa samoilla periaatteilla ja pitkälti samalla mittaushjelmistollakin uudemmissa Android-versioissa. Käytössä olevista Android-laitteista Dalvikin osuus on vielä merkittävä: heinäkuussa 2016 Dalvik oli käytössä noin 50% laitteista [Goo16a].

2.1 Androidin rakenne

Android on mobiilikäyttöjärjestelmä, johon kuuluu oma versionsa Linux-käyttöjärjestelmäytimestä. Siinä on mukana joitakin mobiililaitteille räätälöityjä ominaisuuksia, jotka esimerkiksi vähentävät muistin- ja virrankulutusta [Yag13, s. 49–53].

Android ei muistuta tyypillistä Linux-pohjaista työpöytä- tai palvelinkäyttöjärjestelmää [Yag13, s. 48]. Se on läheisempää sukua riisutummille *upotetuille* Linux-järjestelmille mutta eroaa niistäkin. Ensinnäkin merkittävä osa sovelluksista ja koko järjestelmästä on ohjelmoitu Javalla, ja niitä suoritetaan virtuaalikoneessa. Järjestelmän rakenne ei muutenkaan noudata esimerkiksi POSIX-standardia vaan perustuu hajautettuun komponenttiarkkitehtuuriin, joka on nimeltään *Binder* [Yag13, s. 79]. Arkkitehtuurin infrastruktuuri, virtuaalikone sekä suuri osa järjestelmän matalan tason palveluista on toteutettu C- ja C++-kielillä [Yag13, s. 138].

Kuva 2.1 esittää Android-järjestelmän perusosia. Kuvan arkkitehtuurikerroksista ylimmät ovat lähimpänä käyttäjää ja sovellusohjelmoijaa. Suurin osa Android-sovelluksista ohjelmoidaan Javalla Androidin sovelluskehityksen ohjelmointirajapintaa [Goo16c] käyttäen. Ohjelmointirajapinnan **android**-nimiavaruuden paketit ovat varta vasten Androidia varten rakennettuja. Ne päästävät ohjelmoijan käsiksi käyttöjärjestelmän ja laitteiston palveluihin. Nimiavaruuden **java** ohjelmointirajapinta on osajoukko *Java Standard Edition* -rajapinnasta ja on peräisin *Apache Harmony* -projektista [Yag13, s. 77].



Kuva 2.1: Android-järjestelmän kokonaisarkkitehtuuri [Yag13, muokattu]

Kaikki Java-ohjelmakoodi käännetään Androidia varten ensin normaaliksi tavukoodiksi ja siitä Androidin omaksi rekisteripohjaiseksi **dex**-tavukoodiksi, jota suorittaa Dalvik- tai ART-virtuaalikone [WPC⁺11, s. 15]. Virtuaalikoneessa suoritetaan sekä sovellusohjelmat että järjestelmän sisäiset komponentit – siltä osin kuin ne on toteutettu Javalla.

Android on alusta lähtien suunniteltu tukemaan moniajtoa. Jokaista Android-sovellusta suoritetaan omalla käyttäjätunnuksellaan, omassa Linux-prosessissaan ja omassa virtuaalikoneessaan [Yag13, s. 43–45]. Tällä tavoin sovellukset on eristetty omiin hiekkalaatikoihinsa, ja esimerkiksi sovellusten käyttöoikeuksista voidaan pitää kirjaa käyttöjärjestelmän käyttäjänhallinnan avulla.

Android-ohjelmointirajapinnan luokat antavat sovellusohjelmoijalle yksinkertaisen näkymän käyttöjärjestelmän palveluihin. Valtaosa palveluiden varsinaisesta toteutuksesta sijoittuu kuitenkin Androidin *järjestelmäpalveluihin* (*system services*) eli komponentteihin joita järjestelmä suorittaa omissa palvelinprosesseissaan [Yag13, s. 79].

Yleensä sovellusohjelmoija ei kutsu näitä tukipalveluita suoraan, vaan Android-ohjelmointirajapintojen sisäiset toteutukset kutsuvat niitä Binderin tarjoaman prosessienvälisen kutsumekanismin avulla sovelluksen prosessista käsin. Komponenttien

välinen sidonta on täysin ajonaikaista ja perustuu erilliseen palveluhakemistokomponenttiin.

Osa Androidin järjestelmäpalveluista on toteutettu Java-kielellä ja osa C++-kielellä. Javan natiivirajapinta JNI esitellään tarkemmin luvussa 4, mutta voidaan jo havaita, että natiivirajapintaa käytetään yleisesti järjestelmän sisäisessä toteutuksessa, kun Java-koodista on kutsuttava matalamman arkkitehtuuritason natiiveja kirjastoja [Yag13, s. 78]. Esimerkiksi Binder-kutsumekanismin toteutus on C++-kielinen, mutta Android sisältää myös Java-rajapinnat Binderin käyttöön. Sisäisesti nämä Javan Binder-sidonnat on toteutettu natiivirajapinnan avulla [Ope12]. Niin ikään monet Javalla toteutetut järjestelmäpalvelukomponentit kutsuvat itse natiivikirjastoja JNI:n kautta. Jotkut Android-ohjelmointirajapinnan luokat kutsuvat natiivikirjastoja suoraan JNI:n kautta, mutta ohittavat järjestelmäpalvelutason: esimerkiksi Log-luokka kutsuu suoraan liblog-natiivikirjastoa.

Järjestelmäpalveluita matalammalla olevia arkkitehtuuritasoja, mukaan lukien dynaamisesti ladattavia natiivikirjastoja, ei käsitellä yksityiskohtaisesti. Mainittakoon kuitenkin, että Native Development Kitin [Goo16b] vakaat natiivirajapinnat kuuluvat tälle tasolle arkkitehtuurissa. Näitä dynaamisesti linkitettyjä natiivikirjastoja voi siis kutsua suoraan tavallisilla aliohjelmakutsuilla natiiviohjelmasta käsin ohittaen järjestelmäpalvelukerroksen. Androidin Binder-komponentteihin ei natiiviohjelmoija pääse käsiksi. Natiivityökalujen ulottumattomiin jääkin suurin osa Androidin laajasta toiminnallisuudesta, mutta esimerkiksi peliohjelmointia varten NDK:n natiivirajapinnat ovat riittäviä [Yag13, s. 46–47]. Ne sisältävät muun muassa tuen OpenGL- ja bittikarttagrafiikalle, ääni- ja multimediatoimintoja sekä valmiudet laitteiden sensorien lukemiseen.

2.2 Dalvik-virtuaalikone

Javalla ohjelmoidun Android-sovelluksen matka Dalvikissa suoritettavaksi ohjelmaksi on pääpiirteissään seuraavanlainen. Aluksi sovellus käännetään Java-lähdekoodista tavallisella Java-kääntäjällä luokkatiedostoiksi, jotka sisältävät standardinmukaisen Java-virtuaalikoneen ymmärtämää välikieltä, Java-tavukoodia.

Androidin Dalvik-virtuaalikone ei kuitenkaan ole Java-virtuaalikone. Luokkatiedostoja käyttävä Java-virtuaalikone noudattaa pinopohjaista käskyarkkitehtuuria, jossa konekäskyjen operandit sijaitsevat implisiittisesti virtuaalikoneen pinossa [AS11, s. 15]. Dalvik sen sijaan on rekisteripohjainen virtuaalikone, jossa konekäskyjen ope-

randit viittaavat eksplisiittisesti virtuaalikoneen ylläpitämiin rekistereihin. Lisätietoa virtuaalikoneratkaisuiden eroista antaa lähde [SCEG08].

Androidin kehitystyökalut kääntävätkin ohjelman Java-tavukoodista edelleen *Dalvik Executable* -formaattiin, `.dex`-päätteisiin tiedostoihin. Näistä tiedostoista Dalvik-virtuaalikone lataa ohjelman luokat ja niiden sisältämät Dalvik-käskykannan mukaiset käskyt.

Dalvik-virtuaalikone on *jäljittävä Just-In-Time -kääntäjä (tracing JIT compiler)* [AS11, s. 15-16]. Näin ollen Dalvikin *tulkki* suorittaa Dalvik-käskykannan mukaista ohjelmaa ja pitää samalla kirjaa koodialueista, usein silmukoista, joiden kääntäminen suoraan laitteistoalustan konekielelle nopeuttaisi todennäköisimmin ohjelman suoritusta. Tällaiset ohjelmakohdat Dalvik kääntää konekielelle ja suorittaa konekielisinä. Ne myös tallennetaan JIT-kääntäjän välimuistiin [Yag13, s. 197]. Androidin uudempi ART-virtuaalikone perustuu Java-koodin *Ahead-of-Time* -kääntämiseen (AOT) [Goo16d].

Dalvik on ohjelmoitu C++-kielellä, mutta sen tulkki koostuu pääosin hyvin pitkälle optimoidusta symbolisesta konekielestä, jonka avulla minimoidaan joitakin tulkkien toteutuksille tyypillisiä suorituskykyrasitteita [Ope12].

Jos Dalvikin suoritusajasta toimintaa tarkastellaan Java-ohjelman ulkopuolisesta näkökulmasta, havaitaan, että Dalvik viettää ohjelman suoritustajan hieman yksinkertaistaen joko tulkissa, JIT-kääntäjässä tai jo kääntämänsä metodin konekielisesä ohjelmakoodissa. Lisäksi virtuaalikoneella on toki muita, ylläpidollisia tehtäviä kuten luokkien lataaminen ja muistinhallinta.

Näiden suoritustilojen lisäksi Dalvikin Java-natiivirajapintatoteutuksen ansiosta virtuaalikoneen prosessi saattaa siirtyä suorittamaan valmiiksi käännettyä konekielistä natiiviohjelmaa, palata sieltä väliaikaisesti suorittamaan natiiviohjelman kutsumia Java-ohjelman operaatioita ja palata lopulta takaisin ei-natiiviin suoritustilaansa. Siirtyminen normaalin suoritukseen ja natiivisuoritukseen välillä vaatii virtuaalikoneelta erityistoimenpiteitä esimerkiksi kutsukäytäntöjen yhteensovittamisen takia. Tämän tutkielman aihe on näiden erityistoimien aiheuttamien suorituskykyrasitteiden tarkastelu eli käytännössä Dalvik-virtuaalikoneen suoritusajaisen käyttäytymisen mittaaminen ja tulosten analyysi.

2.3 ART-virtuaalikone

Seuraavaksi käsitellään lyhyesti ART-virtuaalikoneen merkittävimmät erot Dalvikiin verrattuna [Goo16d]. Dalvikin JIT-käännöksen sijaan ART- virtuaalikone kääntää Android-sovelluksen DEX-tavukoodista natiivikoodiksi *kun sovellus asennetaan laitteeseen*. Käännös on siis Ahead-of-Time -kääntämistä.

Toinen keskeinen ero on ART:in tehokkaammassa ja paremmin rinnakkaistuvassa roskienkeruumenetelmässä, johon on kehitteillä myös *tiivistävä* (*compacting*) roskienkeruu [Goo16d]. Viimeksi mainitulla ominaisuudella on suuri merkitys myös natiivirajapinnan käytön kannalta, sillä sen myötä automaattinen muistinhallinta voi siirrellä olioita vapaasti muistissa [Goo16g]. Tutkielman tulosten sovellettavuutta ART-virtuaalikoneeseen käsitellään tutkielman yhteenvedossa.

3 Suorituskyky ohjelmistokehityksessä

Riittävä suorituskyky on keskeinen laatuvaatimus mille tahansa ohjelmistolle, mutta miten suorituskyky tulisi ottaa huomioon ohjelmiston kehitysprosessissa? Ohjelmistotekniikan kirjallisuudessa kysymykseen on vastattu hyvin eri tavoin.

Tämä luku käsittelee sitä, miten ohjelmistolta vaadittava riittävä suorituskyky voidaan saavuttaa ohjelmistoa laadittaessa. Aluksi tutkitaan, missä vaiheessa ohjelmistoprosessia suorituskyky tulisi huomioida. Sitten käsittelyä syvennetään määrittelemällä mittareita, joiden avulla suorituskykyä voi arvioida ja mitata kvantitatiivisesti. Tämän jälkeen käsittely keskittyy konkreettisten ohjelmien suorituskyvyn mittaamenetelmiin ja aivan lopuksi siihen, mitä rajoituksia suorituskykymittausten tuloksia tulkittaessa tulee huomioida.

3.1 Suorituskyvyn hallinta tuotantoprosessissa

Donald Knuthilta on peräisin tunnettu lainaus: "Ennenaikainen optimointi on kaiken pahan alku ja juuri" [Knu74, s. 268]. Knuth korostaa, ettei varsinaisessa ohjelmointityössä kannata käyttää aikaa suorituskykyä parantaviin manuaalisiin optimointeihin, ennen kuin ohjelman suoritusajasta käyttäytymistä on tutkittu ja löydetty ohjelman osat, joiden suoritus aika on merkittävä osa ohjelman kokonaissuoritusajasta.

Periaatteeseen on kaksi syytä. Ensinnäkin ohjelmoijan intuitio johtaa yleensä harhaan, kun etsitään suorituskykyyn eniten vaikuttavia ohjelman osia. Toiseksi äärimmillen optimoitu ohjelmakoodi on vaikeammin ymmärrettävää ja ylläpidettävää, joten on perusteltua jättää suurin osa ohjelmakoodista selkeämmäksi suorituskyvyn kustannuksella.

Knuthin tueksi voi nykynäkökulmasta todeta, että laitteistoarkkitehtuurien, optimoivien kääntäjien, virtuaalikoneiden ja suoritusympäristön yleisen monimutkaisuuden takia pelkästä ohjelman lähdekoodista on yhä vaikeampaa suoraan päätellä ohjelman suoritusajasta käyttäytymistä [HSDH04], [ADMT88].

Knuthin näkökulma rajoittui kuitenkin yksittäisen algoritmin, ohjelman tai komponentin *ohjelmointiin*. Laajaa ohjelmistoa rakentaessa otetaan kantaa myös ohjelmiston käsitteellisiin vaatimuksiin ja suunnitellaan ohjelmiston komponenttitaason arkkitehtuuri sekä komponenttien tarkempi rakenne. Oli ohjelmiston tuotantoprosessi miten inkrementaalinen tahansa, varhaiset suuren mittaluokan ratkaisut

rajoittavat usein myöhempiä yksityiskohtia. Merkittäviä seurannaisvaikutuksia on esimerkiksi käytetyillä ohjelmointikielillä, käytetyillä valmiskomponenteilla ja ohjelmakirjastoilla sekä ohjelmiston yleisarkkitehtuurilla. Esimerkiksi asiakas-palvelin-arkkitehtuurissa on valittava, sijoitetaanko tietty toiminto palvelin- vai asiakaskomponenttiin. Androidin tapauksessa vastaava valinta koskee sitä, sijoitetaanko toiminto hallittuun ympäristöön (Java-ohjelma virtuaalikoneessa) vai natiiviympäristöön (natiivikirjasto).

Kaikkia näitä valintoja voi toki periaatteessa muuttaa milloin tahansa, mutta todellisten projektien rajoitusten takia se on usein jälkikäteen mahdotonta.

Suorituskykyvaatimusten täyttymistä ohjelmistoprosessin näkökulmasta tavoitellaan *ohjelmistojen suorituskykytekniikan* alalla (*Software Performance Engineering, SPE*). Se on ohjelmistotekniikan osa-alue, jossa suorituskykyvaatimukset kvantifioidaan ja otetaan huomioon vaatimusanalyysistä lähtien aina ylläpitoon asti [Smi90, s. 1]. Puutteelliset suorituskykyominaisuudet ovat usein olleet keskeisessä osassa epäonnistuneissa ohjelmistoprojekteissa [Gla97], ja suorituskykytekniikan kehittäjän Connie U. Smithin mukaan ne voitaisiin usein välttää *mallintamalla* ohjelmistojen suorituskykynäkökulmasta jo ennen toteutusvaihetta.

Suorituskykyä käsittelevä kirjallisuus keskittyykin usein joko myöhäisen vaiheen diagnosointiin mittausten avulla tai ohjelmiston mallintamiseen varhaisessa kehitysvaiheessa [WFP07, s. 172]. Mittaamista voidaan tehdä vasta, kun ohjelmisto tai sen osa on jo olemassa. Mallintamista hankaloittaa puolestaan, että käyttökelpoiselta mallilta vaaditaan abstraktisuutta ja yksinkertaisuutta mutta samalla riittävää *tarkkuutta* suhteessa määriteltyn syötejoukkoon sekä reunaehtoihin ollakseen *validi* [Fer78, s. 161–162].

Mittaus- ja mallintamisnäkökulmaa ei ole saatu yhdistettyä toisiinsa riittävän yleispätevästi [WFP07, s. 172]. Tämän tutkielman lähestymistapana on rajata näkökulma spesifisti Java-virtuaalikoneen ja natiivikoodin väliseen viestintään ja konstruoida yksinkertaistettu malli natiivirajapinnasta mittausten perusteella. Tutkielma siis yhdistää omalla tavallaan mittauksen mallintamiseen, sillä mittaustulokset toimivat laadittavan mallin lähtökohtana.

Seuraavaksi käsitellään tutkielmassa käytettyjä suorituskykymittareita yleisellä tasolla, ohjelmiston suorituskyvyn mittaamiseen käytettäviä konkreettisia tekniikoita ja lopuksi itse mittauskohteena olevaa ohjelmistoa sekä mittausten järjestämiseen ja tulosten tulkintaan liittyviä haasteita.

3.2 Suorituskykymittarit

Suorituskyky on pohjimmiltaan subjektiivinen käsite, joka ilmaisee, miten *hyvin* järjestelmä toteuttaa tehtävänsä [Fer78, s. 2]. Tässä tutkielmassa suorituskyky määritellään ohjelmiston laadulliseksi ominaisuudeksi, joka kuvaa, miten *nopeasti* ohjelmisto toimii käyttäjän näkökulmasta: miten tyydyttävät sen vasteajat ovat ja miten tehokkaasti se käsittelee syötedataa.

Ohjelmiston tehokkuus ja nopeus kuvaavat sen käyttäytymistä *ajan suhteen*, mihin myös tutkielma keskittyy. Toinen keskeinen suorituskykymittareiden luokka tutkii ohjelmiston *muistinkäyttöä* [Fer78, s. 457]. Tässä tutkielmassa aikaulottuvuus on ensisijainen ja sitä tutkitaan kvantitatiivisesti; mobiilisovellusten muistinkulutus on toki oleellinen suorituskykytekijä, mutta tutkielmassa käsitellään ainoastaan sen vaikutuksia sovelluksen ajalliseen käyttäytymiseen.

Jotta järjestelmän suorituskykyä voisi arvioida täsmällisesti, käsite on purettava kvantitatiivisiksi *mittareiksi*. Näitä ovat järjestelmän *tuottavuus*, *responsiivisuus* ja *käyttöaste* [Fer78, s. 12]. Tutkielmassa mittari tarkoittaa suorituskyvyn kvantitatiivisia osatekijöitä, oli ne sitten johdettu ohjelmistoa kuvaavista malleista tai saatu todellisista mittauksista.

Tutkielmassa keskitytään mobiilisovellusten responsiivisuuteen. Sitä arvioidaan *vasteajalla*: aikavälillä syötteen saamisesta siihen hetkeen, kun ohjelman tai ohjelma-komponentin tuloste on palautettu käyttäjälle tai kutsuvalle komponentille [Fer78, s. 13]. Vasteajan perussuure on aika. Vasteaikamittauksen mittauskohteen suuruusluokka voi vaihdella jopa yksittäisestä konekäskystä ohjelmiston suorittamaan pitkäkestoiseen, käyttäjälle näkyvään toimintoon.

Myös tuottavuutta käsitellään tutkielmassa, sillä komponenttien tuottavuus vaikuttaa käyttäjän kokemiin vasteaikoihin. Tuottavuus tarkoittaa käsittelytehoa, jolla järjestelmä suoriutuu tietyn työkuorman (*workload*) käsittelystä, ja tuottavuusmittarit ilmaistaan yksiköllä *kuorma aikayksikköä kohti*. Kuormalle ei yleensä ole löydettävissä universaalia, laitteisto- tai sovellusalueriippumatonta mittayksikköä [Fer78, s. 12]. Työkuorman mallintaminen onkin keskeisiä haasteita suorituskyvyn arvioinnissa [Smi90, s. 115], [Fer78, s. 221].

3.3 Mittausmenetelmät

Seuraavaksi käsitellään ohjelmiston vasteaikojen mittaamiseen käytettäviä konkreettisia mittausmenetelmiä, joilla saadaan mitattua kokonaisvasteaikoja sekä vasteaikojen jakautumista ohjelmiston eri osiin.

Vasteajan mittaaminen

Tietyn operaation kokonaisvasteajan mittaaminen on käsitteellisesti yksinkertaista: mitataan aikaväli syötteen saamisesta siihen hetkeen, kun haluttu tulos on tulostettu käyttäjälle. Toisaalta vuorovaikutteisen sovelluksen suorittaman laskennan ja käyttäjän syötteiden vuorottelu voi olla vilkasta. Käyttäjän toiminta on usein samanaikaista järjestelmän laskennan kanssa. Käyttäjän harkinta- ja reagointiviiveet eivät ole osa vasteaikaa, joten vuorovaikutukset on mittausta varten eriteltävä riittävän yksinkertaisiin mitattaviin yksiköihin, joissa käyttäjän ja sovelluksen roolit on erotettavissa [Fer78, s. 14–16], [Smi90, s. 114].

Käyttöjärjestelmän erilaiset keinot mitata aikaa koskevat erilaisia osia kokonaisvasteajasta. Järjestelmän käyttäjä voi mitata sekuntikellolla operaatioihin objektiivisesti kuluva kokonaisaika, *tosiakaa* (*real time*, *wall clock time*) [Fer78, s. 463], [SR08, s. 20]. UNIX-käyttöjärjestelmissä tosiaikaa mittaa järjestelmäkutsu `gettimeofday` [SR08, s. 173], joka lukee laitteiston kelloa. Jos sovellusohjelma kutsuu sitä tavallisena käyttöjärjestelmäkutsuna, saattaa ohjelman suoritus häiriintyä tarkkuutta vaativissa mittauksissa [Fer78, s. 472]. Tässä tutkielmassa kyseistä rutiinia käytetään ainoastaan niin pitkien kokonaisvasteaikojen mittaamiseen, ettei käyttöjärjestelmäkutsujen aiheuttamilla häiriöillä ole merkitystä. Moniajojärjestelmissä tosiaikaa käyttävät mittaukset ovat myös riippuvaisia järjestelmän kokonaistilasta [Fer78, s. 455]; tutkielman mittauksissa muiden prosessien määrä pyritään minimoimaan ja niiden vaikutus tulkitaan mittausten satunnaisvirheeksi.

Kvantitatiivisten mittausten rinnalla kannattaa pitää mielessä käyttäjän subjektiivinen kokemus: käyttäjä voi kokea jopa yhtä suuriksi mitatut vasteajat erilaisiksi, jos käyttöliittymässä näkyvä työn valmistumista havainnollistava edistymispalkki etenee eri tavalla [HAKB07].

Tässä tutkielmassa kaikki suorituskykyä parantavat ratkaisut tavoittelevat ohjelman *vasteaikojen lyhentämistä tosiaikana mitattuna*. Tarkemman suorituskykyanalyysin kannalta pelkkä tosiajan käyttäminen ei kuitenkaan aina ole riittävää. UNIX-käyttöjärjestelmät pitävät kirjaa myös *sovellusajasta* (*user time*) eli ajasta, jonka so-

vellusprosessi viettää aktiivisesti tavallisessa suoritustilassa käyttöjärjestelmäytimen ulkopuolella (*unprivileged mode, user mode*) [SR08, s. 20]. Toinen osa sovellusprosessin suoritusajasta, järjestelmäaika (*system time*), vietetään käyttöjärjestelmäytimen sisällä palveluissa, joita sovellus on eksplisiittisesti kutsunut.

Aika, joka kuluu erilaisia palveluita odottaessa, lasketaan tosiaikaan muttei sovellustai järjestelmäaikaan. Käyttäjän kannalta esimerkiksi I/O-operaatioiden odottelulla on selvästi suuri merkitys. Sovellus- ja järjestelmäaika yhdessä vastaavat sovelluksen *suoritinaikaa* (*CPU time*), koska niihin kuuluu ainoastaan prosessin aktiivinen suoritus aika [SR08, s. 20]. Tutkielmassa oletetaan, että suoritinaika on natiivirajapinnan rasitteiden oleellisin osatekijä, sillä rajapintatoteutus oleellisesti käsittelee keskusmuistissa ja rekistereissä olevaa dataa ja tietorakenteita ja pitää niistä kirjaa. Tosiaikaa mittaamalla varmistetaan kuitenkin, etteivät suoritinajan ulkopuoliset merkittävät viiveet jää huomiotta.

Vasteajan jakautuminen

Kokonaisvasteajalla on merkitystä ohjelmiston käyttäjälle sekä tulosten lopullisessa validoinnissa. Tarkempi suorituskykyanalyysi vaatii kuitenkin tuekseen yksityiskohmaisempia mittauksia. Niiden tavoite on purkaa vasteaika osiin ja tunnistaa ohjelman osat, joiden suorituksella on kokonaisuuden kannalta suurin vaikutus [Fer78, s. 455–456].

Yksittäisen ohjelmiston ajonaikaista käyttäytymistä on käytännöllisintä ja järkevintä mitata ohjelmallisesti eikä erillisillä mittaamiseen käytetyillä laitteistolla. Fyysiset mittauslaitteet ovat tarkkoja ja häiritsevät järjestelmän toimintaa minimaalisesti, mutta laitteiston tasolla tapahtuvien mikrotason tapahtumien havaitseminen on käsitteellisesti kaukana ohjelmiston sisäisestä suorituskontekstista [Fer78, s. 32]. Käytännön ohjelmistokehityksessä mittauslaitteiden hyödyntäminen ei usein ole mahdollista.

Mittaustekniikat jakaantuvat menetelmiin, jotka joko *kirjaavat* (*record*) kiinnostavia *tapahtumia* tai *tarkkailevat* (*monitor*) järjestelmän tai ohjelmiston *tiloja* [Smi90, s. 328]. Tapahtumien kirjaaminen tarkoittaa, että aina kiinnostavan tapahtuman esiintyessä oleellinen informaatio tapahtumasta kerätään ja tallennetaan. Kirjaamistoimet käynnistetään mitattavan ohjelman omasta toimesta, kun taas tarkkailussa mittaamisen kontrolli on ohjelman ulkopuolella esimerkiksi käyttöjärjestelmässä tai laitteistossa. Tarkkailumenetelmät perustuvat yleensä toistuvaan *näytteidenottoon*:

tasaisin väliajoin, satunnaisin väliajoin tai tietyn tapahtuman sattuessa otetaan näyte ohjelmiston tilasta.

Tyypillinen kirjaamismenetelmä on ohjelman *instrumentointi*: suoritettava ohjelma kirjaa mittauksen kohteena olevat tapahtumat eksplisiittisillä, ohjelmaan itseensä sisältyvillä käskyillä [Smi90, s. 328]. Keskeinen esimerkki on ohjelman *kutsuprofilin* rakentaminen laskemalla jokaisen aliohjelman jokainen kutsu. Näin tallennetun informaation avulla voidaan suorituksen jälkeen tulostaa luettelo jokaisesta kutsutusta aliohjelmasta järjestettynä kutsukertojen mukaan. Tämän kaltaista instrumentointia ei välttämättä tarvitse ohjelmoida käsin: kääntäjä voi lisätä instrumentointikäskyjä kohdeohjelmaan tarvittaessa automaattisesti [GKM82, s. 121].

Ohjelman profiilin käsitteen loi Donald Knuth [Knu71]. Alunperin se tarkoitti taulukkoa ohjelman jokaisen lauseen *suorituskerroista*; nykyisessä käytössä ja tässä tutkielmassa profiloinnilla tarkoitetaan yleisemmin sellaista ohjelman suorituskäyttämisen mittaamista, jossa tuloksena on esitys mittausarvojen jakautumisesta ohjelman eri osille, yleensä aliohjelmille. Suorituskerrojen sijaan usein tutkitaan osien suoritukseen käytettyä aikaa. Se hyödyntää usein instrumentoinnin sijaan tai sen lisäksi näytteenottomenetelmiä ja tutkii yleensä ohjelman suoritusaikaa aliohjelmien tarkkuudella.

Näytteenoton tavoite on kerätä ohjelman suoritustiloista edustava otos kirjaamatta jokaista tilanmuutosta ohjelman sisältä käsin [Smi90, s. 328]. Yksittäinen näyte voidaan ottaa esimerkiksi laitteiston tosiaikakellon aiheuttaman keskeytyksen laukaisevana, ja näytteeseen voidaan tallentaa esimerkiksi suorituksessa olleen konekäskyn osoite.

Instrumentoinnin ja näytteenoton avulla ohjelman suoritusaajan jakautumisesta aliohjelmiin tai jopa yksittäisiin konekäskyihin voidaan siis periaatteessa saada hyvinkin tarkkoja mittauksia. Ennen aikaista optimointia loppuun asti vältelleet ohjelmistokehittäjät voivat näin tutkia vaikkapa, missä aliohjelmissa suoritin viettää suurimman osan ajastaan. Algoritmeja muuttamalla, tietorakenteita vaihtamalla tai ohjelmaa muuten muokkaamalla näitä *kuumia kohtia* (*hot spot*) voidaan optimoida – tai vähentää niiden kutsukohtia.

Mittausten toteuttaminen

Instrumentoitua ohjelmaa voi suorittaa normaalisti samassa ympäristössä, jossa tuotantosovellukset tavallisestikin suoritetaan eli Android-laitteessa. Täysipainoinen

näytteenotto sen sijaan vaatii käyttöjärjestelmältä tukea näytteenoton suorittamiselle tiettyjen laitteistokeskeytysten tapahtuessa. Tämä käyttöjärjestelmän toiminto pohjautuu laskureihin ja keskeytyksiin, jotka on varta vasten sisäänrakennettu suorittimiin. Esimerkiksi ARM-suorittimet voi asettaa laskemaan kuluneita suorituskyklejä, väärin ennustettuja suoritushaaroja, muistihakuja sekä muita tapahtumia [ARM10, s. 3-85 – 3-88]. Kun valittu tapahtumamäärä on ylittynyt, keskeytys käynnistää käyttöjärjestelmäytimen mittausrutiinin.

Vaihtoehto normaalin suoritusympäristön käyttämiselle on *virtualisointi*, jossa pelkkä sovellus tai koko ohjelmistoympäristö käyttöjärjestelmineen suoritetaan ohjelmallisessa virtuaalikoneessa. Tällöin virtuaalikoneeseen voi periaatteessa ohjelmoida mitä tahansa räätälöityjä mittauksia.² Virtualisointimenetelmät on rajattu tutkielman ulkopuolelle, sillä tavallisessa Android-laitteessa tehtävät mittaukset antavat kaikki tarvittavat tulokset, ja virtualisoitu suoritusympäristö voi käyttäytymiseltään erota todellisista laitteista tavoilla, joiden toteaminen kuitenkin edellyttäisi mittauksia myös todellisessa ympäristössä.³

Myös Androidin Java-virtuaalikoneessa Dalvikissa on ohjelmien suoritusajasta käyttäytymistä kirjaavia toimintoja, joiden mittauksiin Androidin kehitysympäristön suorituskykytyökalut perustuvat [Ope12]. Dalvik mittaa kuitenkin Java-ohjelman metodien sekä natiivialiohjelmien suoritusajkoja [Goo16f], [CHL⁺12], siinä missä tämän tutkielman tarkoituksena on tutkia Dalvikin itsensä suorituskykyä: Javan natiivirajapinnan toteutus on osa virtuaalikonetta.

Tutkielman mittaukset tehtiin näytteenottotekniikalla, Linuxin *perf*-työkalun avulla [DSZ11, s. 13]. Android-laitteeseen on asennettu uudelleenkonfiguroitu Linux-ydin, joka tukee suorituskykylaskurien käyttöä. Itse mittaukset käynnistetään ja raportit tulostetaan ytimen ulkopuolisilla *perf*-komentorivityökaluilla.

Mittaustekniikoiden valintaperusteista

Eri mitaustekniikoilla on vahvuutensa ja heikkoutensa. Instrumentointi ei vaadi minkäänlaista erityistukea järjestelmältä, mutta edellyttää kuitenkin instrumentoitavien ohjelmien uudelleen kääntämistä ja usein myös niiden muokkaamista käsin.

²Esimerkki virtualisoinnista on Androidin oma *emulaattori*, joka perustuu QEMU-virtuaalikoneeseen. Se on emulaattori, sillä se mallintaa Androidin laitteistoarkkitehtuuria toisella laitteistolla, tavallisella mikrotietokoneella.

³Sovelluksen suorittaminen esimerkiksi Valgrind-virtuaalikoneessa on merkittävästi normaalia hitaampaa.

Käsin tai metaohjelmoinnin avulla lisätyllä instrumentoinnilla saadaan kirjattua yksityiskohtaisinta tietoa ohjelmiston sisäisestä tilasta ja sovellusaluekohtaisesta informaatiosta: esimerkiksi siitä, minkä tyyppiset parametrit ovat yleisimpiä tietyissä kutsuissa tai mihin käyttötapaukseen mitattu tapahtuma liittyy [Smi90, s. 334]. Pelkällä kääntäjän lisäämällä ei-sovelluskohtaisella instrumentaatiolla saadaan myös tietoa, jota ei muilla tekniikoilla tavoita: jokaisen aliohjelman kutsujen määrä ja jokaisen kutsun vasteaika.

Instrumentaatio on tavallaan kattavin mittausmenetelmä, sillä jokaisesta mielenkiintoisesta tapahtumasta saadaan periaatteessa kirjattua tarvittava informaatio. Erityisesti vasteaikojen suhteen ongelmaksi kuitenkin muodostuu se, että instrumentointi aina *häiritsee* enemmän tai vähemmän suoritettavan ohjelman toimintaa [Fer78, s. 44]. Toisin sanoen mittauskohteena onkin instrumentoitu eikä alkuperäinen ohjelma. Jos vasteaikoja mitataan tihein väliajoin käyttöjärjestelmäkutsulla, joka palauttaa järjestelmän kellon arvon, saattaa tämä kutsu ja sen aiheuttama prosessin tilamuutos vaikuttaa ohjelman käyttäytymiseen, vaikka kutsujen suorittamiseen kuluvan ajan vähentäisikin tuloksista.

Toisaalta, jos ohjelmiston suorituskyvyn seuraaminen katsotaan osaksi sen normaalia toimintaa ja jos instrumentointikäskyt ovat oleellinen osa tuotantokäytössä suoritettavaa ohjelmistoa, häirinnän ongelma katoaa. Tämä lienee käytännöllisintä laajoissa palvelinohjelmistoissa tai käyttöjärjestelmissä – yksittäisten käyttäjien mobiililaitteissa suoritettavien sovellusohjelmien jatkuva suorituskykyseuranta ei usein tule kyseeseen, vaan mahdolliset instrumentointikäskyt poistetaan tuotantoversiota rakennettaessa.

Häirinnän ongelma on vähäisempi näytteenottotekniikoissa [Fer78, s. 478], sillä käyttöjärjestelmä keskeyttää ohjelman toiminnan suhteellisen harvoin, ja näytteenoton aiheuttama häirintä jää tilastollisesti pieneksi. Näytteenotossa tutkittavan ohjelmiston tilaa pitää kuitenkin tulkita ohjelmiston itsensä ulkopuolelta, joten käytännössä analyysin pohjana on suorittimen tila näytteenottohetkellä: erityisesti käskyosoittimen (*program counter*) sekä pinon sisällöt [DSZ11, s. 3], [Dun07, s. 4].

Pelkän käskyosoittimen käyttöä suorituskykyanalyysin perustana on kritisoitu [Dun07], [ABL97]. Käskyosoitinnäytteiden avulla saadaan kyllä selville, missä ohjelman osassa suoritin viettää eniten aikaa. Näin on kuitenkin vaikea hahmottaa laajempaa suorituskontekstia, joka selittäisi, mistä eniten suoritusaikaa käyttävää aliohjelmaa on kutsuttu. Suorituskykypullonkaulojen syiden merkityksellisempi analyysi vaatii lähtökohdakseen oikean abstraktiotason.

Tämän vuoksi käyttökelpoinen näytteenotto vaatii kutsupinon uudelleenrakentamista pinon sisällöstä otetuista raakanäytteistä. Nyt aliohjelman A kustannuksiin voidaan laskea mukaan kaikki näytteet, joiden kutsupinoissa A esiintyy, vaikka näytteen aikana suoritin olikin suorittamassa toista aliohjelmää C . Proseduraalisessa ohjelmointiparadigmassa on nimittäin usein mielekästä nähdä aliohjelman A ilmentämä abstraktio myös sellaisen laskennan syynä, joka tapahtuu aliohjelmassa C esimerkiksi kutsuketjun $((A, B), (B, C))$ välityksellä. Yksinkertaisessa yksisäikeisessä ohjelmassa pääohjelman kontrolle laskettaisiin siis *kaikki* suoritus. Käytännössä mielekäs abstraktiotaso näytteiden analyysille löytyy tilannekohtaisesti jostakin pääohjelman ja suoritettavan kohdan väliltä kutsupinosta.

Profilointityökaluissa aliohjelman suoritusaikaa kutsutaan usein *inklusiiviseksi*, jos siihen lasketaan mukaan myös aliohjelman kutsumien muiden aliohjelmien ajat [Goo16f]. Pelkkään käskyosoittimeen perustuva raaka suoritusaika on tässä tutkielmassa *eksklusiivinen* suoritusaika, jota usein kutsutaan nimellä *self time*.

Näytteenotto ei ole mittausmenetelmänä kytketty tarkalleen tiettyihin ohjelmakohtiin. Suorittimen keskeytyksen laukeamisen jälkeen ohjelman suoritus saattaa edetä joitakin konekäskyjä ennen näytteen ottamista [Mis16]. Vaikka periaatteessa työkalut saattavat raportoida jopa yksittäisten konekäskyjen suhteelliset kustannukset, kannattaa näihin suhtautua pienellä varauksella. Tällä epätarkkuudella ei ole käytännön vaikutusta, jos mittauksen kohteena on suurempien kokonaisuuksien kuten kokonaisten aliohjelmien inklusiiviset suoritusaajat.

Tutkielmassa profilointiin käytetään *perf*-työkalun näytteenottotekniikkaa, sillä siinä mittauskohteena on mahdollisimman paljon tuotantoversiota vastaava sovellus. Näytteenoton avulla ei saada selville aliohjelmakutsujen tarkkoja *määriä* vaan ainoastaan niiden suhteelliset osuudet otetuissa näytteissä.

Näytteenoton tilastollinen edustavuus

Kaikenlaisessa näytteenotossa on varmistettava, että otanta on *tilastollisesti edustava*: rajattua *otosta* tutkimalla on voitava tehdä päätelmiä koko *perusjoukosta* [Tri82, s. 469]. Tässä tapauksessa rajallisesta joukosta ajanhetkiä (otos) halutaan päätellä, miten käskyosoittimen ja kutsupinon arvot jakautuvat ohjelman koko suoritusaajalle (perusjoukko). Tämä varmistetaan suoritinsyklien laskemiseen perustuvassa näytteenotossa *systemaattisen otannan* [Coc63, s. 206] menetelmällä. Siinä jokaisella ajanhetkellä on sama todennäköisyys tulla valituksi otantaan, sillä näyte otetaan

systemaattisesti tasaisin väliajoin, ja mittaus aloitetaan satunnaisella ajanhetkellä. Tällöin eri ajanhetkien mittausarvoja ei tarvitse painottaa suhteessa toisiinsa.

Systemaattisen ja jaksollisen näytteenoton suunnittelussa saattaa kuitenkin sattua erilaisia virheitä, joiden seurauksena tietyt ajanhetket valitaan otokseen todennäköisemmin kuin toiset. Jos tutkittavassa prosessissa on jokin jaksollisesti toistuva ilmiö, jonka kanssa näytteenoton jakso sattuu synkronoitumaan, tulokset vääristyvät. Esimerkiksi tutkittavassa järjestelmässä saattaa olla käytössä järjestelmän kelloon perustuvia keskeytyksiä, jotka laukaisevat tiettyjä toimintoja [Fer78, s. 58]. Huonosti valitulla näytteenottojaksolla nämä toiminnot yli- tai alikorostuvat mittauksissa.

Näytteenoton toteutustavassa saattaa myös olla sisäänrakennettuja seurausvaikutuksia, joiden takia mittaus jo lähtökohtaisesti painottaa tiettyjä ajanhetkiä. Koska suorittimen tilaa halutaan tutkia *tosiajassa*, on varmistuttava, että näytteet jakautuvat tasaisesti tosiajan suhteen. Esimerkiksi monet yleisesti käytetyt Java-profilointityökalut kykenevät saamaan näytteitä ainoastaan ohjelman ns. luovutuskohdista (*yield point*), eivät mistä tahansa suorituskohdasta [MDHS10, s. 193].

Tutkielman mittauksissa näytteenotto tapahtuu aina, kun ARM-suorittimen kellosyklejä on tapahtunut ennalta valittu määrä (tapahtuma *cycle count* [ARM10, s. 3:86]). Tietokoneen suorittimen kellotaajuus (sykliä määrä sekunnissa) ei kuitenkaan tyypillisesti ole vakio, vaan mukautuu laskentatarpeeseen [CHCC13] [ARM13, s. 20:7]. Mittauksissa on siis erikseen asetettava suorittimen kellotaajuus vakioksi, etteivät suuren kellotaajuuden ajanhetket ylikorostuisi.

Kannattaa huomata, ettei otannan tarvitse olla satunnainen: systemaattinen jaksollinen otanta riittää, kunhan on varmistuttu siitä, ettei perusjoukko sisällä jaksollisia ilmiöitä [Fer78, s. 58]. Tutkielmassa oletetaan, ettei mitattava järjestelmä sisällä jaksollisia ilmiöitä, joiden jakson pituudella olisi yhteisiä tekijöitä n suoritinsyklin kuluttaman ajanjakson kanssa.

3.4 Mittauskohteen edustavuus ja tulosten yleistettävyys

Vaikka suorituskyvyn eri osatekijöitä voi arvioida ja mitata näennäisen tarkkoilla kvantitatiivisilla mittareilla, tulosten objektiivisuus, toistettavuus ja erityisesti yleistettävyys vaatii mittausjärjestelyn tarkkaa suunnittelua ja tulosten tulkintaa. Yksittäinen suoritusaikamittaus tuottaa mittausarvon tietylle *ohjelma-syöte*-parille tietyllä laitteistolla [Fer78, s. 455], muttei sinänsä kerro mitään yleistä esimerkiksi juuri monikielisten Android-ohjelmien ominaisuuksista. Yksittäiseen mittautulokseen

eniten vaikuttavat kausaaliset syyt saattavat olla kulloisenkin ohjelman, laitteiston tai syötteen sattumanvaraisia ominaispiirteitä ja toteutusyksityiskohtia. Esimerkiksi vasteaika ilmaisee aina tietyn operaation vasteajan tietyllä syötteellä eli kuormituksella, jota on hyvin vaikea spesifioida tavalla, joka olisi riippumaton käsiteltävän järjestelmän ominaisuuksista [Fer78, s. 14].

Nämä seikat vähentävät yksittäisten mittaustulosten hyötyä. Tarkoistakin mittauksista on vaikeaa tehdä sellaisia päätelmiä tulosten syistä, jotka voisi yleistää koskemaan muitakin kuin täsmälleen itse mittaustilanteen kaltaisia tapauksia [SS96, s. 69].

Lisäksi moniajojärjestelmän, kuten Android-järjestelmän, kokonaistila vaikuttaa yksittäisiin mitattuihin vasteaikoihin. Näistä syistä mittariksi tulee ottaa vasteaikojen tilastollinen jakauma tietyn ajanjakson sisällä, jotta ympäristöstä aiheutuvat satunnaisvirheet voidaan ottaa huomioon tilastollisesti [Fer78, s. 14–15]. Silloinkin ohjelman suoritusaikaan perustuvat mittarit ovat riippuvaisia järjestelmästä ja laitteistoalustasta, esimerkiksi mobiililaitemallista ja Android-versiosta [Fer78, s. 454].

Yleispätevyyden puute voi vaikeuttaa mittauksen ja arvioiden tieteellistä arvoa. Käytännön kehitystyössä tämä ei välttämättä ole ongelma, jos tuloksista vedetään vain rajallisia johtopäätöksiä. Kun tavoitteena on tietyn järjestelmän suorituskyvyn *parantaminen*, riittää että mittauksista löydetään suorituskyvyn kannalta ongelmalliset suorituskohdat ja lopullinen suorituskypsykyparannus *validoidaan* vertailemalla alkuperäistä ja muokattua järjestelmää [Fer78, s. 336].

Jos tavoitteena on uuden ohjelmistojärjestelmän tuottaminen suorituskypsykkytekniikan menetelmillä, mittauksia käytetään validoimaan suorituskypsykkyarvioita, jotka on johdettu määrittely- ja suunnitteluvaiheiden malleista [Smi90, s. 18]. Ohjelmistoprosessissa näitä validoituja malleja verrataan lopulta alkuperäisiin suorituskypsykkyvaatimuksiin, joita niitäkin voidaan joskus jopa muuttaa [Smi90, s. 18]. Siten kvantifioitukin mallit ja mittaukset täytyy lopulta suhteuttaa ohjelmiston suorituskypsykkyvaatimuksiin ja viime kädessä käyttäjien subjektiivisiin kokemuksiin.

Vakioidut mittaukset

Eräs tapa saavuttaa astetta yleispätevämpiä tuloksia suorituskypsykkymittauksista on laatia mitattavalle järjestelmälle sarja ennalta laadittuja syötteitä, jotka on suunniteltu edustamaan todellisuudessa kohdattavia syötteitä tai ovat peräisin todellisista työkuormista, ja mitata järjestelmän suoriutumista syötteistä. Kutsun tätä yleistä

menetelmää *vakioiduksi mittaamiseksi (benchmarking)* [VM09, s. 69-71], [CW76], [SS96]. Menetelmää voi hyödyntää, kun tutkittava järjestelmä toteuttaa jonkin spesifikaation tai rajapinnan, joka mahdollistaa samojen testisyötteiden käytön useiden toteutusten kanssa. Tällöin järjestelmän tuloksia voi *verrata* toisten sellaisten järjestelmien tuloksiin, jotka toteuttavat saman spesifikaation. Menetelmää käytetään muidenkin kuin suorituskkyominaisuuksien arvioimiseen. Lisäksi järjestelmien erot erilaisilla syötteillä tulevat näkyviin ja eroista voi päätellä, minkälaisia syötteitä mikäkin järjestelmä prosessoi tehokkaasti.

Esimerkiksi tietyn ohjelmointikielen kääntäjille laadittu vakioitu mittaussarja koostuisi erilaisista tarkkaan valituista käännettävistä ohjelmista. Vastaavalla menetelmällä voisi tutkia joukkoa virtuaalikoneita, esimerkiksi eri JVM-toteutusten suoriutumista JNI-rajapintaa hyödyntävien sovellusten ajamisesta.

Tämän tutkielman kannalta mahdollinen koeasetelma olisi verrata samankaltaisen sovellusohjelman vaihtoehtoisia versioita, jotka kaikki toteuttavat saman spesifikaation eli prosessoivat samanmuotoista dataa. Yksi versio hyödyntäisi laajasti JNI-rajapintaa, toinen olisi puhdas natiiviohjelma ja kolmas puhdas Java-ohjelma. Asetelman etuna olisi ohjelmien realistisuus ja testien kokonaisvaltaisuus - toisaalta esimerkiksi eri tavalla JNI:tä hyödyntävien versioiden tekeminen olisi hyvin työlästä ei-triviaaleille ohjelmille, ja tulokset saattaisivat kertoa enemmän käsillä olevien toteutusten ominaispiirteistä kuin yleispätevästi JNI-rajapinnan käytön vaikutuksesta. Koeasetelma olisi tavallaan analyttinen: valmiiden kokonaisten ohjelmien suorituskkyominaisuudet analysoitaisiin kokonaisuudesta tehtyjen mittausten perusteella.

Alkeisoperaatioiden mittaaminen

Toinen, tässä tutkielmassa käytetty lähtökohta JNI:n suorituskkyominaisuuksien tutkimiseen on ikään kuin synteettinen: JNI-rajapinnan yksittäisten operaatioiden suorituskkyominaisuuksista yritetään päätellä, miten osista rakennettu kokonaisuus mahdollisesti toimisi. Kutsun *alkeisoperaatioiden mittaamiseksi (microbenchmarking)* tällaista menetelmää, jossa verrataan kielen tai järjestelmän yksittäisten perusoperaatioiden rasitteita toisiinsa [SS96, s. 347]. Menetelmän etuna on, että mitaustuloksista saadaan periaatteessa suuri määrä suoraviivaisesti tulkittavaa dataa siitä, mitkä operaatiot ovat suhteessa erityisen raskaita ohjelmistoissa käytettäviksi. Lähestymistavassa on kuitenkin myös vakavia puutteita [Goe05].

Koska kielen tai järjestelmän alkeisoperaatioiden suoritusaika on minimaalinen, käytännössä niiden suorituskykyä mitataan toistamalla operaatioita pitkäkestoisessa silmukassa. Alkeisoperaatioiden mittaaminen (microbenchmarking) on siis vakioitua mittaamista (benchmarking), jossa työkuormana on yhtä operaatiota toistava ohjelma. Koska vakioitujen mittausten työkuormat tulisi suunnitella edustamaan todellisuudessa kohdattavia ohjelmia [VM09, s. 69], ei alkeisoperaatioita mittaamalla tavoitetta selvästikään saavuteta.

Mittaustavan ongelma johtuu siitä, ettei nykyaikaisessa suoritussympäristössä alkeisosista koostuvan ohjelman kokonaissuoritusaikaa voi laskea deterministisesti osien suoritusaajan perusteella [WEE⁺08, s. 5-6]. Todellisuudessa suoritusaika riippuu myös suorittimen sekä ohjelmallisen suoritussympäristön kuten virtuaalikoneen tiloista.

Suorittimessa tämän epädeterministisyyden aiheuttavat välimuistit, käskyliukuhihnat (*pipelines*) sekä kontrollivuon ennustaminen (*branch prediction*) [HLTW03, s. 1038], [WEE⁺08, s. 5-6]. Suorittimen välimuistissa olevan datan käyttö on merkittävästi nopeampaa kuin keskusmuistissa olevan. Se, miten optimaalisesti suoritin onnistuu välimuistiaan täyttämään, riippuu kuitenkin järjestelmän toiminnasta kokonaisuudessaan eikä irrallisista ohjelman osista. Suorittimen käskyliukuhihnat taas mahdollistavat useamman konekäskyn rinnakkaisen suorittamisen – menetelmä edellyttää ohjelman suoritushaarojen ennustamista ja käskyjen aikataulutusta, mitkä toimivat todennäköisesti eri tavalla todellisessa ohjelmassa kuin alkeisoperaatiota mittaavassa silmukassa.

Lisää haasteita kohdataan, kun siirrytään astetta korkeammalle, tarkastelemaan matalan tason kielen kääntäjää. Tyypillisesti mitattavalla alkeisoperaatiolla ei välttämättä ole sivuvaikutuksia, joten kääntäjä saattaa optimointina poistaa mitattavan operaation [Goe05]. Ratkaisuna on poistaa optimoinnit käytöstä mittauksia varten – Java-kääntäjän tapauksessa tämä ei yleensä kuitenkaan ole yksinkertaista, jolloin mitattavaan koodiin on joskus keinotekoisesti lisättävä sivuvaikutuksia.

Korkean tason kielellä tehtävät vakiodutkin mittaukset ovat erityisen virhealttiita, sillä niitä monimutkaistavat *automaattinen muistinhallinta*, *dynaamisen optimoinnin* aiheuttama epädeterminismi sekä *järjestelmän lämmitysviiveet*, jotka johtuvat luokkien lataamisesta ja JIT-kääntämisestä [BMG⁺08, s. 83]. Näiden hallitsemiseksi on omat mittausmenetelmänsä sekä tilastolliset työkalunsa, joita käsitellään tarkemmin luvussa 5.1.

Tutkielmassa mitataan Dalvik-virtuaalikoneen Java-natiivirajapintatoteutuksen eri osien suorituskykyrasituksia suhteessa toisiinsa ja vastaaviin alkeisoperaatioihin

Java- ja C-kielisessä ohjelmassa. Esimerkiksi JNI:n läpi tehtäviä metodikutsuja verrataan tavallisiin aliohjelmakutsuihin Java- ja C-kielissä. Tuloksista johdetaan yksinkertainen malli, joka ennustaa ohjelmiston JNI-operaatioista aiheutuvia rasitteita eri tilanteissa. Mallin parametreina ovat kutsuttu JNI-funktio sekä kutsuissa käytettyjen parametrien määrät ja tyypit.

JNI- ja Java-kutsujen rasitteiden keskinäisestä vertailusta on hyötyä mille tahansa natiivikomponentteja sisältävälle sovellukselle, koska sen avulla voidaan välttää tunnettuja ongelmakohtia komponenttien välisessä kommunikaatiossa. Kysymys, onko natiivirajapinnan käytöstä ylipäättään etua tietyn sovelluksen suorituskyvylle, saa puolestaan lisävalaistusta vasta, kun natiivirajapinnasta koituvat kustannukset suhteutetaan siihen hyötyyn, joka natiivikomponentin käytöstä on verrattuna vastaavaan Java-komponenttiin.

Tutkielman koeasetelma pyrkii parhaansa mukaan ottamaan huomioon Java-ympäristössä tapahtuvan mittaamisen haasteet, mutta jättää osittain lisätutkimuksen aiheeksi, missä määrin mainitut prosessoriarkkitehtuurin ominaisuudet vähentävät tulosten relevanssia todellisten sovellusten näkökulmasta. Tulokset ovat siis suuntaa antavia, mutta lisätukea tulosten oikeellisuudelle antavat kuitenkin seuraavat huomiot.

1. Pääasiallisena mittauskohteena on C++-kielellä ja konekielellä toteutetun Dalvik-virtuaalikoneen sisäiset osat, joihin virtuaalikoneen kääntäjän suorittamat optimoinnit eivät ulotu. Erityisesti kääntäjä ei voi optimoida JNI-kutsuja pois Java-koodista, koska sillä ei ole tietoa natiivikomponentin mahdollisesti aiheuttamista sivuvaikutuksista. Vertailukohteena mitattavan puhtaan Java-koodin oikeellisuus pyritään tarkistamaan virtuaalikoneen kääntäjän välimuistin konekäskyjä tutkimalla.
2. Koska JNI:tä käyttävät ohjelmakohdat rasittavat käytännössä pelkästään pienenköä käskyaluetta eli Dalvikin JNI-toteutusta, voidaan arvioida, että suorituksen välimuistien ja ennakoivan heuristiikan kannalta käyttötapaukset edustavat JNI:n käyttöä tehokkaimmillaan, sillä koodin ja datan lokaalisuus on suuri. Saadut tulokset voidaan JNI:n osalta tulkita optimistisiksi, mikä huomioidaan tulosten arvioinnissa.
3. Koska mittauksissa yhdistetään kokonaisvasteaikojen mittaaminen profilointiin, saadaan JNI-toteutuksen suoritusajaisesta käyttäytymisestä yksityiskoh-

taista tietoa mallien pohjaksi, jolloin voidaan päätellä, mitkä JNI:n osat ovat suorituskyvyn kannalta kriittisimpiä.

4. Mittauksissa alkeisoperaatioille annettavia mahdollisia parametreja vaihdellaan hallitusti. Yksittäisten epävarmojen vasteaika-arvojen sijaan saadaan kuvaajia, joissa vasteaika vaihtelee halutun muuttujan funktiona. Mittausten oikeellisuudelle saadaan varmistusta regressioanalyysin avulla, sikäli kuin mittauksissa on näkyvissä selviä korrelaatioita. Yhdistettynä profilointiin menetelmällä saadaan joka tapauksessa mielenkiintoista tietoa JNI-rajapinnan toiminnasta vähintäänkin kvalitatiivisella tasolla, vaikka numeeristen mittausarvojen relevanssi jää epävarmaksi.

Tutkielman analyysivaiheessa esitetään ehdotuksia täydentävistä mittauksista ja vaihtoehtoisista tavoista saada luotettavampia tuloksia tutkimuskohteesta. Tutkielman tavoitteena on hyödyntää mittaustuloksia siten, että niitä tulkitsemalla voidaan laatia malli, jota voisi hyödyntää ohjelmistojen suorituskykytekniikan [Smi90] menetelmien osana ja ensi askeleena kehittyneempään suorituskykymalliin.

4 Java-natiivirajapinta

Kaikki Android-järjestelmään asennettavat tavalliset sovellukset suoritetaan omassa prosessissaan virtuaalikoneessa, vaikka ne sisältäisivätkin natiivikomponentteja. Tutkimusten mukaan [LLDW11] [LJ10] [BSS⁺09] natiivikomponenttien käytöstä saattaa joskus todella olla suorituskykyetua: koska Java-ohjelmaa suoritetaan välikielelle käännettynä virtuaalikoneessa, seuraa tästä yleisrasitteita verrattuna suoraan konekielellä suoritettavaan ohjelmaan. Kuinka siis korkean tason kielestä on mahdollista siirtyä suorittamaan konekielistä ohjelmaa?

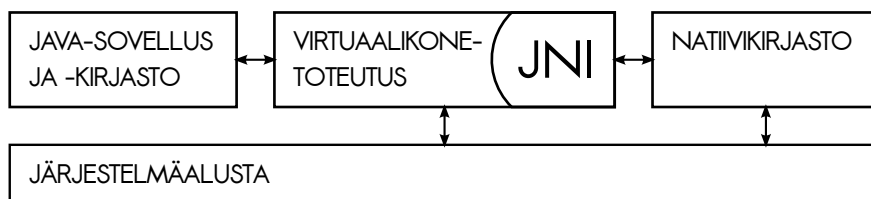
Tämä luku käsittelee Java Native Interface -rajapintaa. Aluksi JNI-rajapinta ja vieraskutsurajapinnat esitellään yleisesti. Sen jälkeen tutkitaan suunnitteluperiaatteita ja perusongelmia, joiden vuoksi JNI-rajapinnan suunnitteluratkaisuihin on päädytty. Rajapintaa esitellään myös sovellusohjelmoijan näkökulmasta. Lopuksi käsitellään tarkemmin vieraskutsurajapintojen keskeistä ongelmakohtaa eli muistinhallintaa JNI-rajapinnan näkökulmasta.

4.1 Java Native Interface ja vieraskutsurajapinnat

Vieraskutsurajapinta (*Foreign Function Interface*) on rajapinta, jonka kautta korkean tason ohjelmointikielestä käsin voidaan kutsua matalan tason kielellä ohjelmoituja aliohjelmia [HG07]. Java Native Interface on tällaisen vieraskutsurajapinnan spesifikaatio [Lia99].

Minimaalisin mahdollinen vieraskutsurajapinta mahdollistaa matalan tason aliohjelmien kutsumisen ja mekanismin, jolla korkean tason datan esitysmuodot voidaan muuttaa matalan tason esitysmuotoihin ja päinvastoin [RS06, s. 49]. JNI tarjoaa lisäksi takaisinkutsumekanismin, jonka avulla natiiviohjelmasta voi kutsua Java-ohjelman metodeja sekä käsitellä Java-ohjelman dataa [Lia99, s. 41].

Kuva 4.1 antaa kokonaiskäsitteen JNI:n roolista. Sovelluksen kokonaisuuden kannalta Javan natiivirajapinta tukee kahdenlaista integraatiota natiivikomponentin ja virtuaalikoneen välillä. Ensimmäisessä vaihtoehdossa suorituksen pääkontrolli on Java-kielisellä sovelluksella, joka kutsuu ajoittain *natiivikirjaston* tarjoamia palveluita [Lia99, s. 5]. Toisessa vaihtoehdossa pääsovellus toteutetaan matalan tason kielellä ja sovellukseen *upotetaan* Java-virtuaalikone, jonka avulla sovellus voi suorittaa Java-kielisiä ohjelmia.



Kuva 4.1: Natiivirajapinnan asema [Lia99, s. 5]

Android-sovellukset ovat lähtökohtaisesti Java-kielisiä, joten ainoastaan ensimmäinen integraatiomekanismi on käytettävissä sovellusohjelmoijalle. Natiivikomponentit toteutetaan siis *dynaamisesti linkitettävänä kirjastoina*, jotka virtuaalikone lataa Java-ohjelman käyttöön.

4.2 Natiivirajapinnan suunnitteluperiaatteet

Javan natiivirajapinnan suunnittelussa on tavoiteltu tasapainoa kahden keskenään ristiriitaisen vaatimuksen kanssa [Lia99, s. 145–146]. Toinen on *binääriyhteensopivuus*: saman binäärimuotoisen natiivikirjaston tulee toimia eri virtuaalikoneiden kanssa ilman uudelleenkääntämistä. Tämän vuoksi JNI ei voi tehdä mitään oletuksia virtuaalikoneen sisäisestä toteutuksesta. Periaate on tärkeä ennen kaikkea siksi, että virtuaalikoneen on aina kyettävä järjestämään uudelleen omia tietorakenteitaan esimerkiksi automaattisen muistinhallinnan, JIT-kääntämisen ja erilaisten optimointien toteuttamiseksi. JNI:tä edeltänyt varhainen Javan natiivirajapintatoteutus oli suoraviivaisempi ja asettikin rajoituksia virtuaalikoneen roskienkeruualgoritmeille [Lia99, s. 8].

Yhteensopivuuden vaatimus on ristiriidassa toisen keskeisen vaatimuksen, *hyvän suorituskyvyn* kanssa. Yhteensopivuus on mahdollista saavuttaa ainoastaan piilottamalla virtuaalikoneen tietorakenteet natiiviohjelmalta rajapinnan taakse. Sen sijaan, että C-kieliselle natiiviohjelmalle välitettäisiin esimerkiksi suoraan osoitin Java-taulukon toteutukseen, natiiviohjelma lukee taulukon alkioita aina natiivirajapinnan apufunktioiden kautta. Yksinkertaisen taulukkotietorakenteenkin käsittely natiivikoodissa edellyttää siis useita kutsuja JNI:n tarjoaman rajapinnan kautta.

Yhdessä suhteessa Javan natiivirajapinta ei ole toteutusriippumaton: se ei kykene estämään synkronointiongelmia tilanteessa, jossa virtuaalikone ja natiivikomponentti käyttävät eri säietoteutuksia [Lia99, s. 97–98]. Onnistunut rajapinnan ylittävä säikei-

den koordinointi edellyttääkin, että virtuaalikoneen säikeet käyttävät perustanaan natiivialustan säikeitä. Dalvikin tapauksessa näin onkin [Goo16e]; kannattaa huomata, että natiivimetodit suoritetaan aina samassa virtuaalikoneen prosessissa ja säikeessä, joka natiivikutsun suorittaa.

4.3 Natiivirajapinta sovellusohjelmoijan näkökulmasta

Java-sovellus voi JNI:n avulla kutsua C- tai C++-kielellä toteutettuja natiiveja aliohjelmia ja vastaavasti natiiviohjelma voi kutsua Java-ohjelman metodeita ja käyttää sen tietorakenteita.

Java-kieli tarjoaa ohjelmoijalle

- varatun sanan **native**, jolla voi esitellä *natiivimetodeja* antamatta niille toteutusta sekä
- järjestelmäkutsun `System.loadLibrary`, jolla ladataan natiivikirjasto, joka sisältää natiivimetodien toteutukset.

Näitä mekanismeja käytettyään ohjelmoija voi kutsua natiivimetodeja aivan kuten tavallisiakin metodeja. Natiivimetodien parametrien ja paluuarvojen tyyppinä on käytettävä Java-tyyppejä.

C-funktioita edustavat natiivimetodit ovat natiivikoodin *ainoita* kielen tasolla tuettuja vastinpareja Java-ohjelmassa. Vaikkapa C-kielen tietueita tai C++-kielen oliioita ei voi käsitellä ja välittää arvoina tai osoitinarvoina Java-kielessä. Eräs tapa säilyttää Java-ohjelmassa viite natiivitietorakenteeseen onkin sen osoitteen tallentaminen `long`-kenttään; osoitteen tulkitseminen osoitteeksi on tietenkin mahdollista ainoastaan natiivikoodissa [Lia99, s. 123–130].

C-ohjelmoijalle virtuaalikone tarjoaa laajan rajapinnan, jonka funktioiden avulla C-ohjelmoija voi monipuolisesti käsitellä Java-ohjelman tietorakenteita. Rajapinta myös määrittelee, mitä C-kielen primitiivityyppejä Javan primitiivityypit vastaavat, joten natiivimetodille välitetyjä primitiivityyppisiä parametreja voi C-kielessä käsitellä suoraan.

Lisäksi merkittävä osa JNI-rajapinnasta koostuu erilaisista viitteiden ja nimien hallintaan liittyvistä apufunktioista, joita tarvitaan kielten erilaisten suoritussympäristöjen yhteensovittamiseksi.

Natiivirajapinnan käyttö C++-kielellä ei oleellisesti eroa sen käytöstä C-kielellä eikä suorituskyykyeroa tapausten välillä ole, sillä molemmat käyttävät täsmälleen samaa virtuaalikoneen JNI-toteutusta [Lia99, s. 106]. Tutkielmassa käsitellään rajapinnan C-kielistä versiota.

Natiivimetodien käyttöönnotto

Käytännössä natiivikirjasto otetaan Java-sovelluksen käyttöön seuraavasti. Natiivimethodi esitellään Java-kielessä luokan sisällä, kuten tavallista, mutta metodin määreeksi lisätään varattu sana **native**. Esittely riittää, eikä metodille anneta toteutusta Java-luokan sisällä. Seuraava esimerkki kuvaa tilannetta [Lia99, s. 13].

```
class NativeExample {
    private native void print(int num); ❶

    public static void main(String[] args) {
        new NativeExample().print(5);
    }
    static {
        System.loadLibrary("native_example"); ❷
    }
}
```

Natiivimetodin **print** ❶ toteutus kuuluu kirjastoon **native_example**, joka on ladattava erillisellä **System.loadLibrary** -kutsulla ❷. Tässä lataaminen tapahtuu *staat-tisessa alustuslohkossa* luokan **NativeExample** lataamisen yhteydessä.

Javan käännöstyökalut tulostavat natiivimetodeja sisältävän luokkamäärittelyn perusteella C-otsaketiedoston, joka sisältää natiivimetodia vastaavan C-kielisen funktion esittelyn. Tästä huolehtii komento **javah -jni NativeExample**, joka ottaa syötteen luokkatiedoston. Natiivimetodia **print** vastaavan funktion esittely näyttää C-kielessä seuraavalta.

```
JNIEXPORT void JNICALL
Java_NativeExample_print (JNIEnv *, jobject, jint);
```

Funktiomäärittelyn **JNIEXPORT**- ja **JNICALL** -makrot piilottavat natiivimetoditoteutusten kutsu- ja linkityskäytäntöjen mahdolliset yksityiskohdat [Lia99, s. 170]. Automaattisesti generoitu metodinimi **Java_NativeExample_print** takaa, että virtuaalikone osaa kytkeä toteutuksen **NativeExample**-luokan **print**-natiivimetodiin. Jos tätä nimeämiskäytäntöä ei halua noudattaa tai toteutuksia täytyy voida vaihtaa ajonaikaisesti, voi natiivimetodeita rekisteröidä joustavammin rajapinnan tarjoamalla **RegisterNatives**-funktiolla [Lia99, s. 101–102].

Funktion `print` kutsuparametreista ensimmäinen, `JNIEnv *`-tyyppinen parametri välitetään kaikkiin natiivimetodeihin. Se on osoitin, josta on pääsy virtuaalikoneen suorituskontekstia edustavaan tietueeseen, joka myös sisältää natiivirajapinnan kaikki palvelut funktio-osoittimien muodossa.

Koska `print` on ilmentymämetodi, on toisen parametrin tyyppi `object`. Parametri vastaa Java-kielen ilmentymämetodien implisiittistä `this`-viitettä luokan kulloiseenkin ilmentymään. Luokkametodin tapauksessa natiivifunktio saisi vastaavasti luokkaa edustavan `jclass`-tyyppisen parametrin. Loput funktion parametreista ovat vastinpareja natiivimetodille määritellyille parametreille: `jint` vastaa Javan primitiivityyppiä `int`.

Lopuksi natiivifunktiolle on vielä kirjoitettava toteutus C- tai C++-kielellä ja käännettävä toteutuskoodi osaksi dynaamisesti ladattavaa kirjastoa. Androidin tapauksessa tämä käännosvaihe edellyttää *ristiinkääntämistä* (*cross compiling*), koska kohdelaitteiston ARM-suoritin ja suoritussympäristö eroavat kehitysympäristön vastavista. Käännösosvaiheesta yksityiskohtineen huolehtivat Androidin Native Development Kit -rakennusskriptit komentorivikäskyllä `ndk-build`. Lopputulos on Android-sovelluksen mukaan paketoitava natiivikirjasto, jonka sisältämää natiivimetodia voi kutsua Java-ohjelmasta aivan kuten tavallista Java-metodia.

Natiivimetodin esittelyn lisäksi kaikki varsinainen JNI-ohjelmointi tapahtuu natiivikielen puolella. Seuraavaksi esitellään oleelliset tietotyypit ja palvelut, joiden avulla natiivimetodit toteutetaan.

Tietotyypit

C-kielestä on voitava käsitellä kaikkia Javan tietotyypppejä. Javan tyypit kuvautuvat C- ja C++-tyypeille taulukon 4.1 mukaisesti [Lia99, s. 166]. C-kieliset tyyppinimet ovat peräisin `jni.h`-otsaketiedostosta. Natiivimetodien primitiiviparametrien arvot ja primitiivityyppiset paluuarvot välitetään natiivirajapinnan yli kopioimalla.

Viitetyyppiset parametrit sitä vastoin välitetään kopioimalla `object`-tyyppinen osoitinparametri. Parametri ei osoita suoraan virtuaalikoneen oliotietorakenteeseen, vaan oliota *edustavaan*, JNI-toteutuksen hallitsemaan tietorakenteeseen. Tällaista osoitinparametria voi käyttää ainoastaan välittämällä se parametriksi JNI:n apufunktioille – muistipaikan arvoa, johon osoitin viittaa, ei koskaan suoraan noudeta tai käsitellä natiivikomponentissa.

Taulukko 4.1: Tyypit

Java	C	bittejä	etumerkki
boolean	jboolean	8	ei
byte	jbyte	8	kyllä
char	jchar	16	ei
short	jshort	16	kyllä
int	jint, jsize	32	kyllä
long	jlong	64	kyllä
float	jfloat	32	—
double	jdouble	64	—
Object	jobject	?	—

Oliot ja luokat

Olioiden ja luokkien kenttien käsittely ja metodien kutsuminen muistuttaa epä-suoruudessaan Java-kielen reflektiorajapintaa. Esimerkiksi olion ilmentymämetodia kutsutaan kolmessa vaiheessa seuraavasti.

Aluksi haetaan viite olion luokkaan funktiolla `GetObjectClass`:

```
jclass GetObjectClass(JNIEnv *env, jobject obj);
```

Sitten luokasta haetaan metodin tunniste metodin nimen ja tyyppin perusteella funktiolla `GetMethodID`.

```
jmethodID  
GetMethodID(JNIEnv *env, jclass clazz, const char *name, const char *signature);
```

Etsittävän metodin tyyppin ilmaisee merkkijonoparametri `signature`, joka noudattaa JVM-tyyppisyntaksia [Lia99, s. 48]. Esimerkiksi kokonaisluvun palauttavan ja kaksi merkkijonoa parametreinaan ottavan metodin tyyppimääritys on `(Ljava/lang/StringLjava/lang/String)I`.

Vasta lopuksi metodia varsinaisesti *kutsutaan* funktiolla `CallτMethod`. Funktiosta on oma versionsa jokaiselle mahdolliselle paluuarvon tyyppille, joten varsinaisen funktion nimi saadaan korvaamalla symboli τ taulukon 4.1 ensimmäisen sarakkeen sisällöllä. Esimerkiksi kokonaisluvun palauttavaa Java-metodia kutsutaan seuraavalla funktiolla.

```
jint CallIntMethod(JNIEnv *env, jobject obj, jmethodID methodID, ...);
```

Vastaava prosessi vaaditaan olion kenttien läpikäymiseen.

Kokonaisuudessaan `sum`-nimisen Java-metodin kutsuminen natiivimetodista käsin voisi näyttää seuraavalta:

```
#include <jni.h>

JNIEXPORT void JNICALL
Java_CallBackExample_nativemethod (JNIEnv *env, jobject receiver_object, jint num) {

    jclass receiver_class = (*env)->GetObjectClass(env, receiver_object);

    jmethodID sum_method_id =
        (*env)->GetMethodID(env, receiver_class, "sum", "(II)I");

    jint result =
        (*env)->CallIntMethod(env, receiver_class, sum_method_id, num, 5);

    ...
}
```

Metodin tai kentän etsiminen symbolisen nimen ja tyyppimäärittelyn perusteella `GetMethodID`-kutsulla on raskas operaatio käytettäväksi toistuvasti silmukassa [Lia99, s. 56–57]. Siksi natiiviohjelmaa suositellaan säilyttämään metodien ja kenttien tunnisteet omissa muuttujissaan, kun ne on kerran selvitetty. Paras käytäntö on tehdä `GetMethodID`-kutsut erillisessä natiivimetodissa, jota kutsutaan sen Java-luokan staattisessa alustuslohkossa, jonka metodeja natiivikomponentti tulevaisuudessa kutsuu [Lia99, s. 56]. Virtuaalikone takaa, että luokan staattinen alustuslohko suoritetaan, ennen kuin luokan metodeja voi kutsua.

```
class InstanceMethodCall {
    private static native void initIDs(); ❸
    private native void nativeMethod(); ❶
    private void callback() { ❷
        System.out.println("In Java");
    }
    public static void main(String args[]) {
        InstanceMethodCall c = new InstanceMethodCall();
        c.nativeMethod();
    }
    static {
        System.loadLibrary("InstanceMethodCall");
        initIDs(); ❹
    }
}
```

Edeltävässä esimerkissä [Lia99, s. 56] natiivimetodista ❶ kutsutaan Java-metodia ❷. Tätä ennen tunnus selvitetään ja tallennetaan valmiiksi normaalilla `GetMethodID`-kutsulla natiivimetodissa ❸, jota kutsutaan staattisessa alustuslohkossa ❹.

Alkuperäisessä natiivirajapinnan spesifikaatiossa arvioidaan, että tunnisteiden tallentamisesta huolimatta takaisinkutsurajapinnan käyttäminen on tyypillisissä toteutuksissa hitaampaa kuin natiivimetodien kutsuminen. Javasta juuri funktiokutsujen epäsuoruuden takia ja siksi, ettei tätä käyttötapaa yleensä ole optimoitu [Lia99, s. 58]. Dalvikin tapauksessa todelliset suorituskykytulokset tulevat nähtäväksi mitausten myötä.

Merkkijonot ja taulukot

Java-merkkijonoja sekä -taulukoita käytetään natiivikomponentista erikseen niiden käsittelyyn tarkoitettujen `JNIEnv`-rajapintafunktioiden kautta. Näistä erityisfunktioista on kahdenlaisia versioita. Toiset kopioivat halutun määrän taulukon alkioita (tai merkkijonon merkkejä) Java-tietorakenteen sisältä natiivimuistialueeseen, kun taas toiset palauttavat natiivimetodille osoittimen virtuaalikoneen hallitsemaan yhtenäiseen muistialueeseen, jota natiivikomponentti voi suoraan käsitellä. Eri menetelmien reunaehdot suorituskyvyn suhteen eroavat toisistaan [Lia99, s. 24–40]. Seuraavassa näitä eroja käsitellään lyhyesti merkkijono-operaatioiden kautta.

C-kielinen ohjelma saa *osoittimen* 16-bittisen Unicode-merkkijonon sisältöön seuraavalla funktiolla.

```
const jchar* GetStringChars(JNIEnv* env, jstring string, jboolean* is_copy);
```

Parametri `jstring string` on merkkijonoviite, joka on aiemmin välitetty natiivifunktiolle natiivimetodikutsun parametrina.

Vaikka `GetStringChars` palauttaa osoittimen, JNI-spesifikaatio kuitenkin sallii virtuaalikoneen luoda merkkijonon sisällöstä uuden kopion ja palauttaa osoittimen siihen. Jos näin tapahtui, `GetStringChars`-funktio välittää `is_copy`-osoittimen kautta `jboolean`-muuttujaan arvon `JNI_TRUE`.

Merkkijonoresurssi on aina lopuksi vapautettava eksplisiittisellä `ReleaseStringChars`-kutsulla:

```
void ReleaseStringChars(JNIEnv* env, jstring string, jchar *cstr);
```

Dalvik-virtuaalikone tukee olioiden *kiinnikyttemistä* (*pinning*), joka estää olion muistiosoitteen muuttumisen roskienkeruun aikana [Ope12]. Periaatteessa tämän pitäisi mahdollistaa nopeat osoittimiin perustuvat merkkijono- ja taulukko-operaatiot, jotka eivät siis Dalvikissa suorita kopiointia eivätkä aiheuta muistinvaraus- tai kopiointikustannuksia.

Android-dokumentaation mukaan kopiointikustannuksia syntyy lähinnä, jos Dalvikin sisäisestä 16-bittisestä Unicode-merkkijonototeutuksesta siirrytään UTF-8 -koodattuun merkkijonoon [Goo16e]; lähes kaikista JNI-merkkijonofunktioista on sekä Unicode- että UTF-8-versiot (esimerkiksi `GetStringUTFChars` on UTF-8-versio `GetStringChars`-funktioista). Dalvik-virtuaalikoneen natiivirajapinnan merkkijono-operaatioiden suorituskyvyn tarkempi analyysi vaatii kuitenkin mittauksia ja virtuaalikoneen toteutuksen tutkimista.

4.4 Muistinhallinta natiiviohjelmoinnissa

Kuten merkkijono-operaatioista huomataan, natiivirajapinnan perushaaste on sovittaa yhteen osapuolten erilaiset muistinhallintamenetelmät. Java-komponentti hyödyntää virtuaalikoneen automaattista roskienkeruuta, kun taas natiivikomponentin muistinhallinta on manuaalista. Tämä aiheuttaisi ongelmia, jos toisella puolella rajapintaa allokoitun tietorakenteen rajallinen elinkaari estäisi toisella puolella tapahtuvan rakenteen käsittelyn.

Primitiivityyppisten parametrien arvot välitetään rajapinnan yli kopioimalla, joten ongelmaksi jäävät viitteet allokoituihin rakenteisiin. Viitetyypiset parametrit kuten taulukot ja olioinstanssit välitetään natiivikomponentille JNI-osoitintyyppien muodossa [Lia99, s. 23]. Jos Java-ohjelmaan ei jää viitteitä olioon, josta välitetään viite natiivipuolelle, saattaisi virtuaalikoneen roskienkeräys poistaa olion. JNI:ssä tämä on estetty: oletuksena oliosta välitetään *paikallinen viite* (*local reference*), jonka voimassaolo taataan natiivimetodin kutsun ajaksi. Viitearvojen pidempiaikainen säilyttäminen edellyttää natiiviohjelmoijalta eksplisiittistä *globaalien* viitteiden luomista ja vapauttamista natiivirajapinnan funktioiden avulla.

Koska JNI-spesifikaatio ei ota kantaa virtuaalikoneen sisäiseen toteutukseen, se ei myöskään sanele, miten virtuaalikoneen tulee toteuttaa spesifikaation määräämät takuut tietorakenteiden elinkaarille. Dalvik-virtuaalikoneen muistinhallinta tukee allokoitujen kohteiden kiinnikytkemistä eli sen takaamista, ettei niiden muistialueita siirretään muistissa osoitteesta toiseen [Ope12]. JNI-spesifikaatio sallii kuitenkin virtuaalikoneen myös *kopioida* kohteen arvon natiivikomponenttia varten. Tutkimme myöhemmässä vaiheessa, millaisia suorituskykyrasitteita kiinnikytkemisellä on kopiointiin verrattuna.

Viitteiden hallinta ja olioiden elinkaari

Erityistapauksissa natiiviohjelmoinnin muistinhallinta edellyttää natiiviohjelman sisältämien erityyppisten Java-viitteiden eksplisiittistä hallintaa. Natiivirajapinta tarjoaa natiivikomponentille kolmentyyppisiä viitteitä virtuaalikoneen olioihin: *paikallisia viitteitä*, *globaaleja viitteitä* ja *heikkoja globaaleja viitteitä* (*local references*, *global references*, *weak global references*).

Kuten mainittu, JNI:n palauttavat suorat osoittimet virtuaalikoneen merkkijonoihin ja taulukoihin tulee aina eksplisiittisesti myös vapauttaa, jotta niiden virtuaalikoneessa käyttämät muistialueet voidaan vapauttaa. Oletuksena kaikki muut olio-viitteet, jotka JNI antaa natiiviohjelman käyttöön, ovat paikallisia viitteitä: niitä ei yleensä tarvitse manuaalisesti vapauttaa, sillä niiden elinkaari on automaattisesti sidottu natiivimetodin kutsun alkamiseen ja päättymiseen [Lia99, s. 62]. Niiden käyttö ei ole säieturvallista. Globaalit viitteet on eksplisiittisesti luotava `NewGlobalRef`-kutsulla; ne estävät olion roskienkeruun ja mahdollistavat olioihin viittaamisen yli eri natiivimetodien kutsujen myös eri säikeistä.

JNI:n ohjelmoijalta edellyttämä eksplisiittinen viitteidenhallinta ja virtuaalikoneen varaamasta muistista huolehtiminen tarkoittaa, että ohjelmoija on vastuussa muistinkäytön tehokkuudesta ja muistivuotojen välttämisestä. Erilaisilla viitteidenhallintastrategioilla on potentiaalisesti myös erilaisia rasitteita, jotka näkyvät vasteajoissa. Virtuaalikoneen suorittama roskienkeruukin on osa ohjelman suoritusaikaa, ja käytetyt viitetyypit vaikuttavat siihen, milloin roskienkeruuta voidaan suorittaa.

Paikalliset viitteet eivät vaadi roskienkeruuta, mutta sisältävät nekin omat rasitteensa. JNI-spesifikaatio tarjoaa niiden eksplisiittiseen hallintaan funktioparin `PushLocalFrame` ja `PopLocalFrame`, joita väitetään tehokkaaksi tavaksi hallita lokaaleja viitteitä useampi viite kerrallaan [Lia99, s. 68]. Yhden natiivimetodin kutsuhan voi siirtää ohjelman suorituksen pitkäksi aikaa syvälle natiivikomponenttiin, joten kyseinen metodikutsu saattaa pitää paikallisten viitteiden edellyttämät muistivaraukset käytössä hyvinkin pitkään, ellei viitteitä erikseen vapauteta.

Ennakoin, että natiivirajapinnan ylittäminen voi aiheuttaa ylimääräisiä suorituskykyrasitteita aivan tavallisissa laskentatehtävissä kuten merkkijonojen ja taulukoiden käsittelyssä, olioiden kenttien ja metodien käyttämisessä sekä natiivimetodien kutsumisessa Java-ohjelmasta käsin. Rasitteet voivat aiheutua Java- ja natiivialiohjelmien erilaisista kutsukäytännöistä, menetelmistä kuroa umpeen eroja natiivikielten ja virtuaalikoneen muistinhallinnassa, operaatioiden edellyttämästä rajapintakut-

sujen määristä sekä JNI-operaatiossa toistuvasta osoittimien käytöstä. Seuraavaksi eri natiivioperaatioiden suorituskykyä mitataan eri parametreilla ja tuloksista luodaan malli, joka auttaa hahmottamaan tapoja Javan natiivirajapinnan tehokkaalle hyödyntämiselle Android-sovelluksissa.

5 Suorituskykymittaukset

Tutkielman aineisto saatiin mittaamalla JNI-operaatioita käyttäviä ohjelmia Android-laitteessa ja analysoimalla tuloksia. Tässä luvussa kuvaillaan mittausten koeasetelma, esitellään mittaustulokset ja rakennetaan niiden avulla parametrisoitu malli, joka ennustaa rajapinnan suorituskykykäyttäytymistä.

5.1 Koeasetelman kuvaus

Tutkielmaa varten laadittiin alkeisoperaatioiden mittausohjelmisto (*microbenchmarking suite*) *NativeBenchmark*. Ohjelmiston avulla Android-laitteessa voidaan suorittaa kokoelma *testejä* eli minimaalisia aliohjelmia, jotka yhdessä käyttävät kattavasti JNI-rajapinnan eri kutsuja.

Kun alkeisoperaatioita suoritetaan laitteessa, samojen operaatioiden suoritusta mitataan kahdella toisiaan täydentävällä menetelmällä eri mittausajojen aikana: ensin yksittäisen aliohjelman suorituksen kokonaisvasteaika mitataan Javan `System.nanoTime()`-metodilla, ja seuraavassa ajossa samoille aliohjelmille luodaan näytteenotomenetelmällä kutsuprofiilit Linuxin `perf`-työkalun avulla.

Mittausten tavoitteet

Yksittäisen funktiokutsun mitattu vasteaika ei kerro mitään yleispätevää kyseisestä funktiosta (ks. luku 3.4, s.18). Siksi samaa funktiota kutsutaan vaihdellen eri kutsukerroilla kaikkia sellaisia asioita, joita funktiokutsussa voi muuttaa: käytännössä kutsuparametrien määrää ja niiden tyyppiä. Jos kyseessä on merkkijonoja, taulukoita tai muita vaihtelevankokoisia syötteitä käsittelevä funktio, vaihdellaan käsiteltävän syötteen kokoa.

Esimerkiksi JNI-funktio `CallVoidMethod` vastaanottaa vaihtelevan määrän kutsuparametreja. Mittausajoissa sitä kutsutaan eri määrällä eri tyyppisiä parametreja, jolloin saadaan selville muun muassa, käyttäytyykö funktio eri tavalla käsitellessään primitiivityyppejä ja viitetyyppejä parametreja. Tämän ansiosta mittauksista piirtyy hieman kokonaisvaltaisempi kuva siitä, mitkä mittausten muuttujat ovat oleellisia funktion suorituskyvyn kannalta.

Toinen tapa lisätä tulosten hyödynnettävyyttä on seuraava periaate: jos jollekin JNI-kutsulle on keksitty analoginen kutsu tai operaatio JNI-rajapinnan ulkopuolel-

ta – operaatio, jonka voi toteuttaa puhtaasti C- tai Java-kielisessä ohjelmassa – lisää mittauksiin tällaiset testit. Siten esimerkiksi JNI-rajapinnan `CallVoidMethod`-perusfunktio-kutsun kaikille variaatioille on laadittu vertailukohdaksi testit, joissa C-ohjelma kutsuu aivan tavallista `void`-paluuarvollaista funktiota, ja Java-ohjelma vastaavasti kutsuu `void`-paluuarvollaista metodia.

Kolmas tapa saada syvempiä tuloksia JNI-rajapinnan käyttäytymisestä on mitata samoista testeistä myös kutsuprofiilit. Kun vasteaikamittauksista on löytynyt mielenkiintoisia riippuvuuksia muuttujien ja vasteaikojen väliltä, nähdään saman testin kutsuprofileista usein selvästi, mikä JNI-toteutuksen osa aiheuttaa esimerkiksi lineaarisesti lisääntyvän vasteajan, kun vaikkapa viitearvoisten kutsuparametrien määrää lisätään.

Toteutuksen haasteet ja reunaehdot

Erilaisten kutsuvariaatioiden kokonaismäärä on suuri, eikä variaatioita voi yleensä toteuttaa ajonaikaisina muutoksina ohjelman syötteissä: esimerkiksi metodiparametrien määrä on Java-ohjelmassa oltava tiedossa käännösaikana. Variaatioista on vieläpä parhaimmillaan luotava 4 eri versiota:

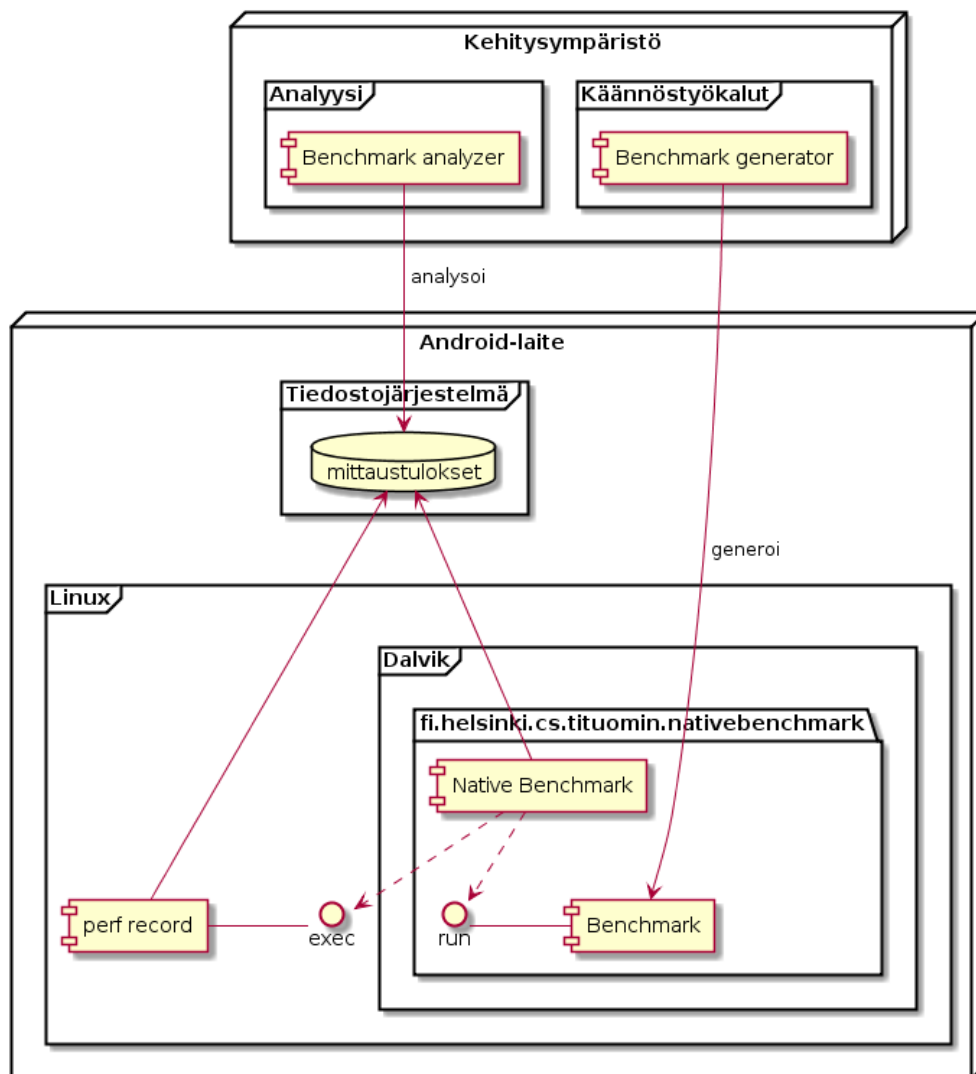
1. C-kielinen versio,
2. Java-kielinen versio,
3. Java-ohjelma, joka kutsuu natiivimetodia, sekä
4. C-ohjelma, joka kutsuu Java-metodia.

Variaatioiden hallitsemiseksi on hyödynnetään yksinkertaista staattista metaohjelmointia: testejä ei ohjelmoida käsin, vaan niiden lähdekoodi generoidaan tietorakenteista, joihin on kuvattu halutut tyyppivariaatiot.

Koska Java-kieli ei suoraan tue staattisen metaohjelmoinnin mahdollisuutta, eikä C++-kielen geneerisiä malleja voisi hyödyntää kuin natiiviversioiden ohjelmointiin, päätettiin lähdekoodin generointi tehdä täysin erillisellä Python-kielisellä ohjelmalla. Seuraavassa hahmotellaan tästä perusratkaisusta syntynyttä arkkitehtuuria.

Ohjelmiston kokonaisarkkitehtuuri

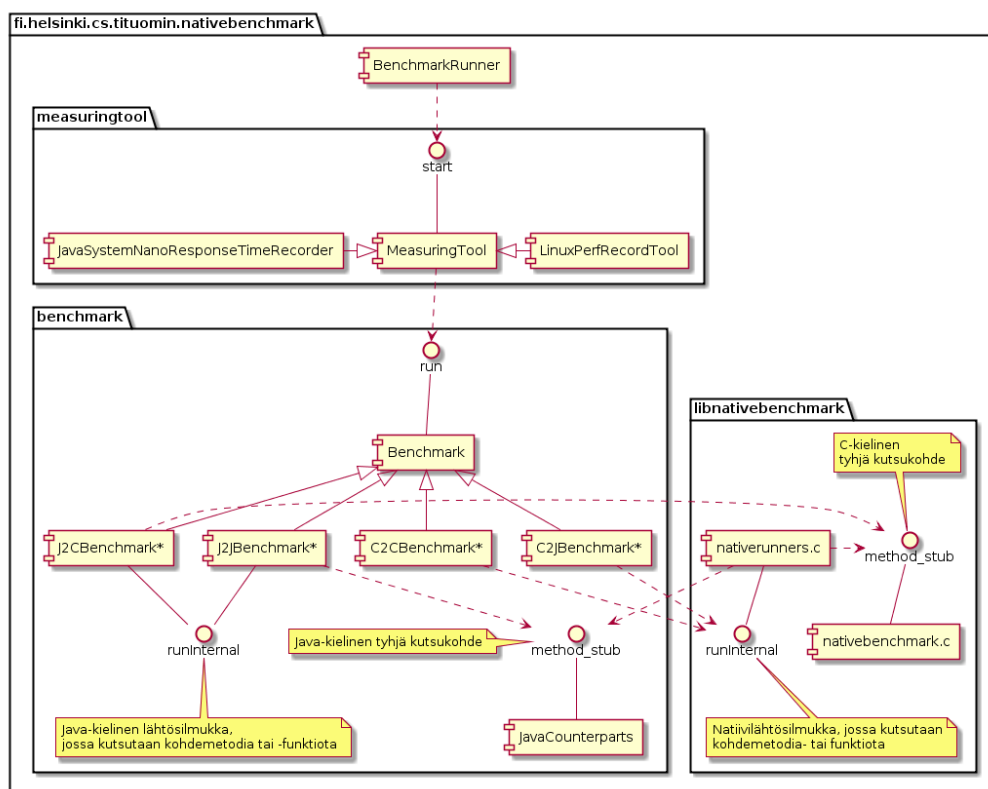
Ohjelmisto koostuu kolmesta komponentista, joista käännöstyökalut ja analyysityökalut ajetaan kehitysympäristössä työpöytä-Linux-järjestelmässä ja mittaustyökalut Android-laitteessa. Ohjelmiston pääkomponentit näkyvät kuvassa 5.1, ja ne käsitellään seuraavassa tarkemmin.



Kuva 5.1: Arkkitehtuurin yleiskuvaus

Android-sovellus NativeBenchmark

Android-sovellus **NativeBenchmark** sisältää kaiken Android-laitteessa suoritettavan ohjelmakoodin: Java-kielisten ja natiivikirjastoon kuuluvien testien lisäksi sovelluk-



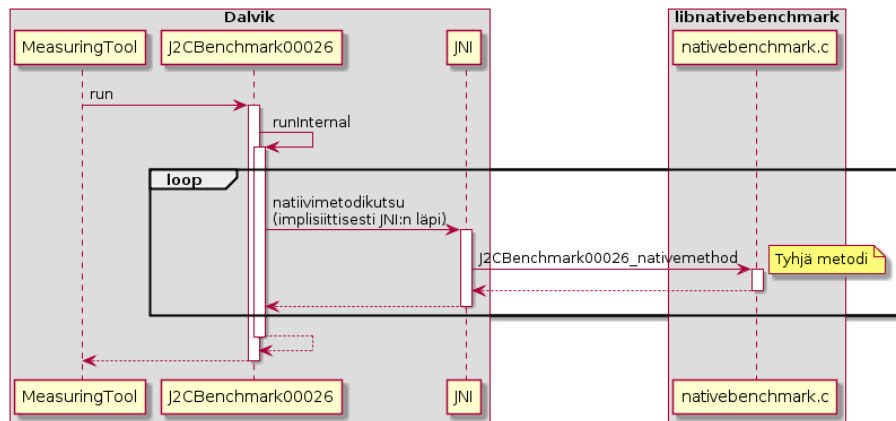
Kuva 5.2: NativeBenchmark

seen kuuluu ohjelmalogiikka, joka suorittaa halutut testit valituilla mittaustyökaluilla.

NativeBenchmark huolehtii mittausajoja koskevien asetustiedostojen lukemisesta ja tulostaa kaikki mittaustulokset ja niihin liittyvän metadatan tiedostoihin. Sovelluksessa on yksinkertainen graafinen Android-käyttöliittymä, mutta sovellusta voi ohjata myös laitteen ulkopuolelta tekstikomentojen avulla TCP-yhteyden välityksellä.

Kuvan 5.2 komponenttikaaviossa näkyvät sovelluksen keskeisimmät osat:

- `benchmark` -paketin sisältämät generoidut testit,
- `libnativebenchmark` -natiivikirjasto, joka sisältää generoitujen testien natiiviosat,
- `BenchmarkRunner` -luokka,
- `MeasuringTool` -luokan kaksi aliluokkaa `JavaSystemNanoResponseTimeRecorder` sekä `LinuxPerfRecordTool`.



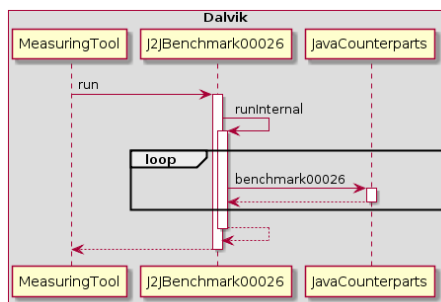
Kuva 5.3: Javasta kutsutaan natiivikomponenttia

Testit ovat pohjimmiltaan vain monta kertaa toistettavia silmukoita (*lähtösilmukoita*), joiden sisällä kutsutaan tutkinnan kohteena olevaa JNI-palvelua tai sen verrokkia. Jokainen testi on ympäröity samanrakenteisella Java-kielisellä **BenchMark**-luokan aliluokalla — silloinkin, kun kyseessä on sellainen verrokkioperaatio, joka suoritetaan puhtaasti **libnativebenchmark**-natiivikirjaston sisällä. Tällaisessakin tapauksessa mittauksen kuitenkin aina käynnistää Java-komponentin **Benchmark Runner**-olio.

Mittauskohteet voi luokitella kahteen joukkoon: funktiokutsuihin sekä muihin JNI:n tarjoamiin palveluihin. Ensiksi mainittu joukko koostuu yksinkertaisista funktiokutsuista JNI-rajapinnan kautta: C-koodissa kutsuista **CallMethod** -funktioihin ja Java-koodissa kutsuista **native** -määreellä merkittyihin **nativemethod** -nimisiin metodeihin. Näille funktiokutsuille on myös verrokkioperaatiot, joissa Java-ohjelma kutsuu suoraan Java-metodia ja C-ohjelma C-funktiota. Toinen joukko testaa JNI-rajapinnan muita palveluita kuten merkkijonojen ja taulukoiden käsittelyä.

Seuraavaksi havainnollistetaan ensimmäisen joukon kaikkia neljää permutaatiota, jotka saadaan lähtösilmukoiden ja kutsukohteiden toteutuskielistä: (Java→C), (Java→Java), (C→Java) sekä (C→C). Aluksi käsitellään kutsu Javasta C:hen.

Kuvan 5.3 sekvenssikaaviossa lähtösilmukka on Java-komponentissa (metodissa **J2CBenchmark00026.runInternal**). Lähtösilmukkaa toistettaessa ylitetään hyvin monta kertaa JNI-rajapinta ja kutsutaan sen kautta **libnativebenchmark** -natiivikirjastossa olevaa metodia, jonka sisältö on tyhjä. Näin ollen mittauksen kohde on käytännössä juuri JNI-kutsu.

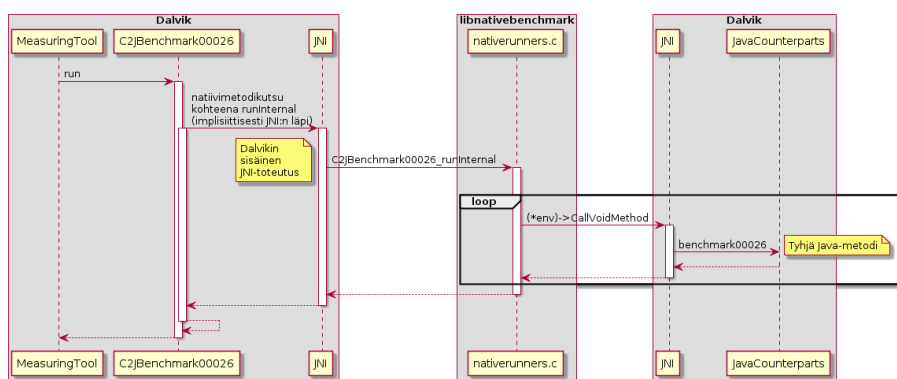


Kuva 5.4: Javasta kutsutaan Java-metodia

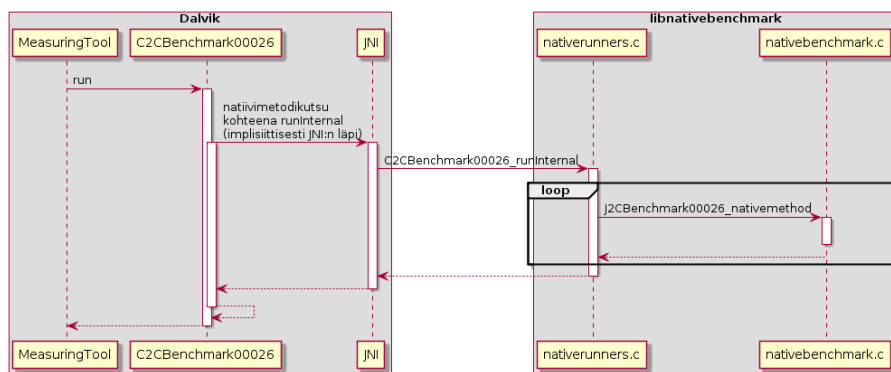
Kuvassa 5.4 nähdään yksinkertaisempi asetelma, jossa Java-ohjelma kutsuu sisäisesti tavallista Java-metodia ilman JNI-rajapintaa. `JavaCounterparts`-luokka sisältää kaikki tyhjät metodit, joita voidaan kutsua Java- tai C-lähtösilmukasta. Samaa luokkaa hyödynnetään myös seuraavassa tapauksessa, jossa C-lähtösilmukka kutsuu Java-tynkämetodia.

Kuvassa 5.5 JNI-rajapinta ylitetään kahdessa kohtaa. Koska tällä kertaa lähtösilmukka on natiivikoodissa (moduulin `nativerunners.c` funktiossa `runInternal`), täytyy jo tätä funktiota kutsua JNI:n kautta. Varsinainen mittauskohde on kuitenkin natiivikoodin kutsuma `(*env)->CallVoidMethod` -JNI-funktio, jota kautta kutsutaan Java-komponentissa olevaa `JavaCounterparts`-luokan tyhjää metodia. Molemmat kuvan Dalvik-virtuaalikoneet ovat sama instanssi, joka on selkeyden vuoksi piirretty kaksi kertaa.

Kuvan 5.6 tilanne on viimeinen permutaatio. Siinä koko mitattava suoritus tapahtuu natiivikomponentissa.



Kuva 5.5: Natiivikomponentista kutsutaan Java-metodia



Kuva 5.6: Natiivikomponentista kutsutaan natiivikomponenttia

Mittauslaitteisto

Mittaukset suoritettiin Samsungin valmistamassa *Google Nexus S*-puhelimessa, joka julkaistiin vuonna 2010. Puhelimessa on Androidin versio 4.1.2. Perf-työkalun käytön mahdollistamiseksi puhelimeen käännettiin uudelleen konfiguroitu Linuxin ydin, ja Android-järjestelmän pakkauksiin otettiin mukaan perf-ohjelma sekä uudelleen käännetyt versiot Dalvikista ja su-komennosta. Profilointi vaatii myös profiloitavien komponenttien kuten Dalvikin uudelleenkääntämistä erikseen määritellyllä kutsukonventiolla, jotta perf-työkalu osaa tulkita aliohjelmakutsujen muodostamat ketjut oikein kutsupinosta otetuista näytteistä.

Mikään käytetyissä mittausmenetelmissä ei estä mittausten toistamista uusilla Android-laitteilla, mikä olisikin tulosten toistettavuuden ja yleistettävyyden kannalta tärkeä seuraava askel. Mittausohjelmiston siirtäminen on helpointa Android Open Source Projectin (AOSP) suoraan tukemiin kohdelaitteisiin kuten *Nexus*-laitteisiin. Lisäominaisuudet, joita mittausohjelmisto edellyttää Android-alustalta, on toteutettu suoraan muokkaamalla ja konfiguroimalla AOSP:n julkaisemaa lähdekoodiversiota, ja AOSP:n lähdekoodiversiot kääntyvät suoraan ilman lisätyötä vain tuetuille laitteille. Avoimen lähdekoodien komponenttien lisäksi Android vaatii nimittäin toimiakseen binäärimuotoisia laitteistokohtaisia kirjastoja, jotka AOSP on julkaissut vain tuetuille laitteille suoraan osana laitteistokohtaisia julkaisuversioita [aos16]. Samoin tuettujen laitteistojen tarkat konfiguraatiot kääntämistä varten ovat osa AOSP-lähdekoodia. Kääntäminen onnistuu kuitenkin mille tahansa laitteelle, jonka binäärikirjastot ja käännöskonfiguraatiot ovat saatavilla.

Oleellisin tutkielmassa avoimeksi jäävä kysymys on, miten Dalvikin korvaava ART-virtuaalikonetoteutus muuttaa tuloksia. Sen takia mittaukset olisi hyvä ajaa vähin-

tään Android 5.0-järjestelmässä. Koska JNI on standardoitu rajapinta, mittauskoodissa itsessään ei ole mitään riippuvuuksia juuri Dalvikiin. Mittaussovelluksessa ei myöskään ole merkittäviä riippuvuuksia käytettyyn Android-versioon. Suurin työmittausten kääntämiselle uudemmille alustaversioille onkin juuri Linux-ytimen sekä Androidin käyttöjärjestelmäkomponenttien uudelleenkonfigurointi tukemaan `perf`-työkalua.

Mittaamisen haasteita

Mittauksien käynnistämistä monimutkaistaa hieman se, että eri `MeasuringTool`-mittaustyökalut suorittavat samat kohdeoperaatiot eri tavoilla.

Yksinkertaisempi tapaus on vasteaikojen mittaus `JavaSystemNanoResponseTimeRecorder`-luokassa: lähtösilmukkaa suoritetaan yhteensä aina täsmälleen ennalta määrätyn kierroslukumäärän verran. `JavaSystemNanoResponseTimeRecorder` ottaa talteen järjestelmän kellon arvon ennen kierrosten suoritusta ja sen jälkeen.

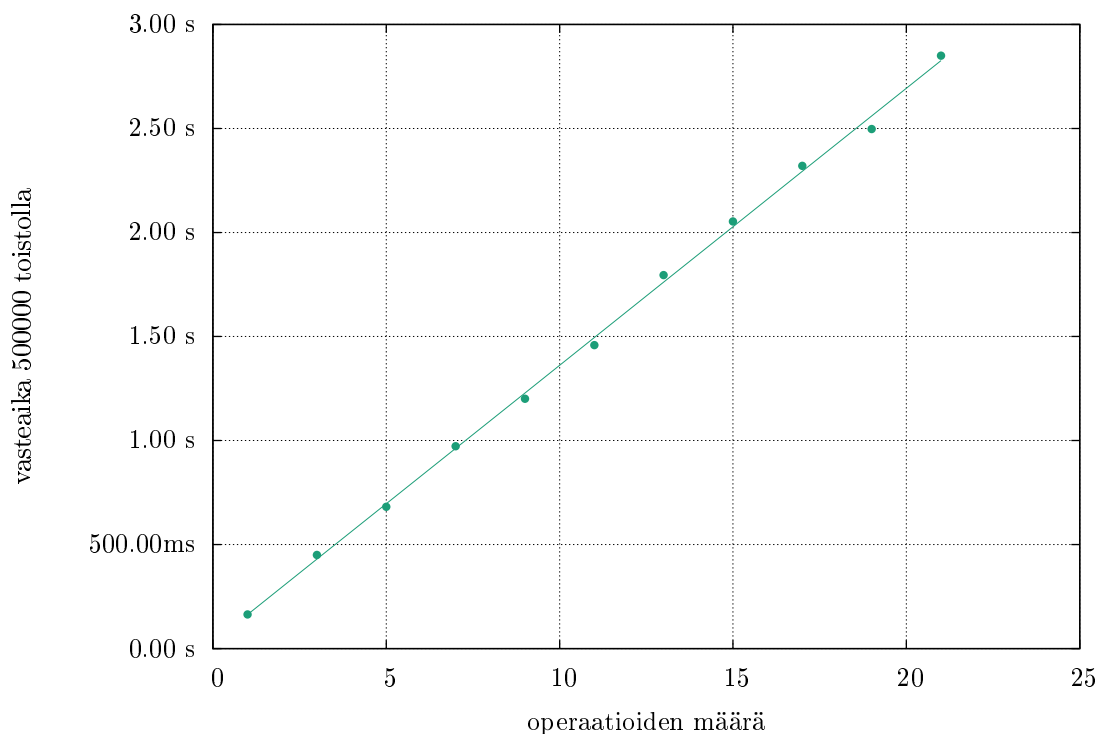
Kutsuprofilia luotaessa kaikkia lähtösilmukoita tulee sen sijaan suorittaa niin kauan, että `perf record`-työkalu on saanut kerättyä riittävän määrän näytteitä – esimerkiksi 10 sekunnin ajan jokaista testiä kohden. Näin ollen lähtösilmukoita ajetaan, kunnes mittaava komponentti `LinuxPerfRecordTool` keskeyttää testisäikeen suorituksen. Tämä monimutkaistaa lähtösilmukoita, sillä niiden on itse tutkittava, onko mittaus keskeytetty, mutta tehtävä se tarpeeksi harvoin, jotta se ei vaikuta suoritusaikoihin.

Mittaamisen perusrasitteet

Eri permutaatioiden sekvenssikaavioista (kuvat 5.3–5.6) voi huomata, että mitattava ohjelmakoodi vaihtelee sen mukaan, onko `runInternal`-metodi ja lähtösilmukka Java- vai natiivikomponentissa. Kun se on Java-komponentissa, Dalvik ensin tulkitsee silmukkaa ja sitten JIT-kääntää sen suorittaakseen kääntämänsä versiota silmukasta. Kun lähtösilmukka on natiivikomponentissa, C-kääntäjän tuottamaa konekielistä versiota suoritetaan sellaisenaan.

Koska on mahdotonta kontrolloida, miten erilaisia kaksi eri versiota lähtösilmukasta konekielitasolla ovat, ei niiden suoritusaikojakaan periaatteessa voi suoraan vertailla. Lähtösilmukoiden erojen mittaamiseksi molempien versioiden perusrasitetta arvioitiin seuraavalla menetelmällä. Kummastakin lähtösilmukasta luotiin testisarjat, jossa suoraan silmukan sisälle lisätään n kappaletta identtistä ohjelmalohkoa, jotka

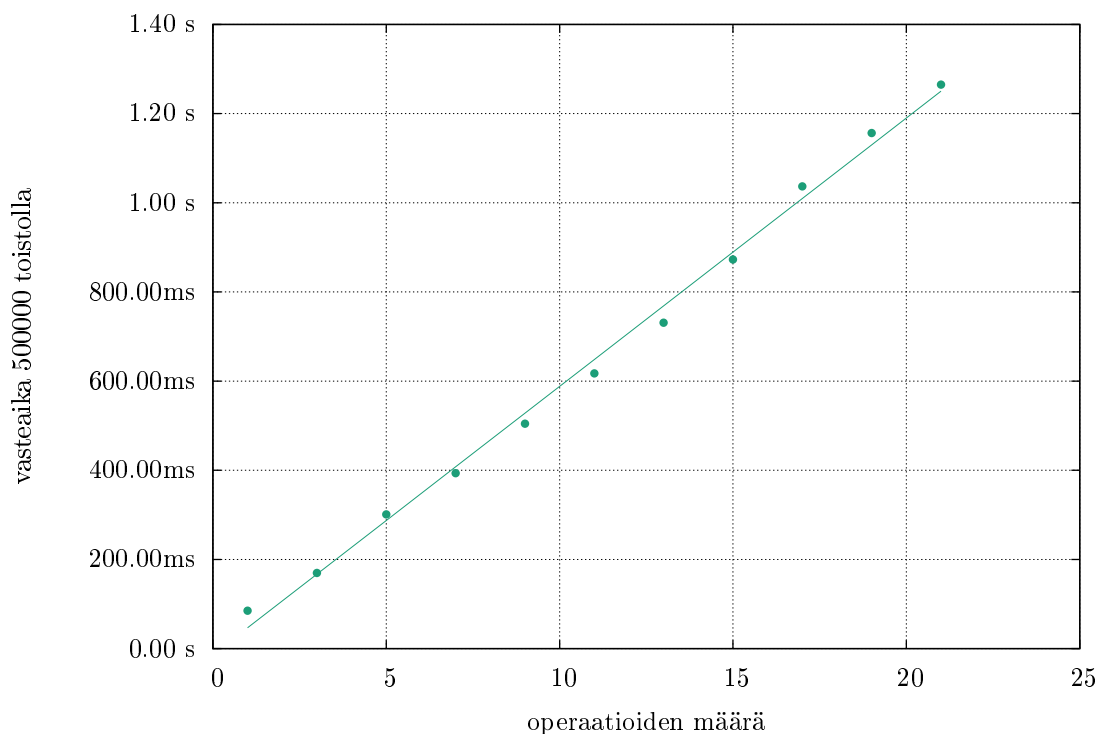
Kuva 5.7: Mittauksen perusrasite (C)



suorittavat hyödyttömiä mutta tarpeeksi raskaita ja sivuvaikutuksellisia aritmeettisiä operaatioita paikallisilla muuttujilla. Kun $n:n$ arvoja käydään läpi lineaarisesti, saaduista vasteajoista interpoloimalla voidaan arvioida, mikä mittausmenetelmän pohjavasteaika on $n:n$ arvolla 0 (tyhjä lähtösilmukka). Näin saadaan tarkempi arvio perusrasitteesta kuin pelkästään ajamalla tyhjää lähtösilmukkaa – erityisesti Java-koodin tapauksessa Java- tai JIT-kääntäjä saattaa myös optimoida pois tyhjän silmukan, jolla ei ole sivuvaikutuksia.

Perusrasitemittauksen tulokset näkyvät C-kieliselle lähtösilmukalle kuvassa 5.7 ja Java-kieliselle kuvassa 5.8. Käytännön mittauksissa osoittautui kuitenkin, että ainakaan Java-koodissa menetelmä ei toimi, sillä perusrasitteeksi interpoloinnissa saadaan negatiivinen arvo. Vaatisi lisätutkimuksia kehittää perusrasitemittauksista käyttökelpoisia. Ongelma saattaa liittyä näissä mittauksissa käytettyihin epätyypillisiin ohjelmalohkoihin, joiden käyttäytyminen JIT-käännöksen syötteenä ei ole ennustettavaa.

Kuva 5.8: Mittauksen perusrasite (Java)

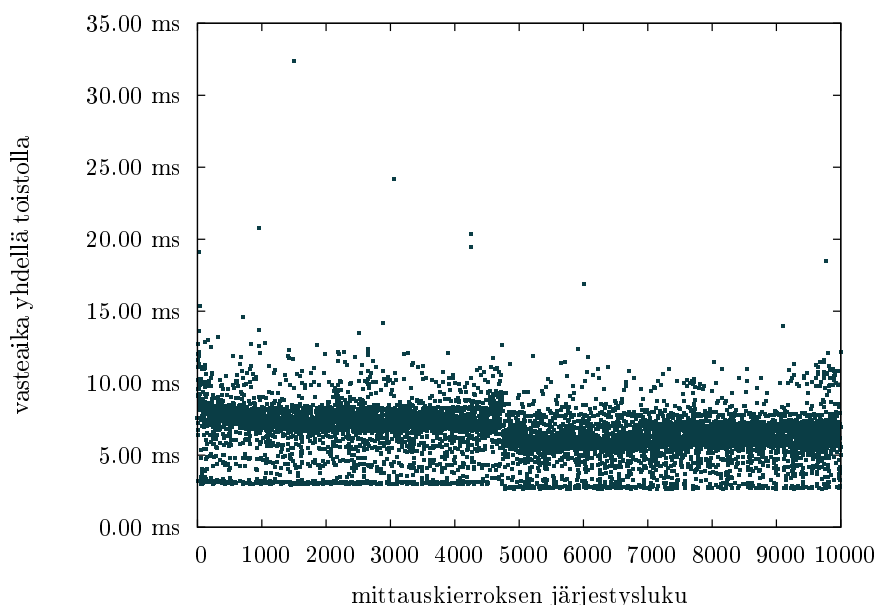


Muistia allokoivat operaatiot

Muistia allokoivia JNI-operaatioita, jotka esimerkiksi luovat olioita, ei voi suorittaa tuhansia kertoja ilman, että sovelluksen muisti loppuu. Yleensä mittauksissa on pyritty pitämään kierrosmäärä niin suurena kuin käytännössä on järkevää, jotta satunnaisten järjestelmässä esiintyvien häiriöiden suuruus verrattuna mitattuihin kokonaisvasteaikoihin olisi pieni. Allokoivissa operaatioissa silmukan kierrosmäärä pidetään alhaisena, mutta vastaavasti samoja mittauksia suoritetaan useampia ajoja ja saaduista luvuista otetaan keskiarvo.

Javan tapauksessa väliaikaisten olioiden allokointi aiheuttaa lisäksi roskienkeruuta ennakoimattomina ajankohtina. Mittauksissa roskienkeruun aiheuttama sinänsä oleellinen rasite suorituskyyvylle on päätetty yksinkertaisuuden vuoksi jättää huomiotta. Roskienkeruuta ehdotetaan virtuaalikoneelle `System.gc()` -kutsulla mitausajojen ulkopuolella. Androidin lokien perusteella virtuaalikone myös näyttää käytännössä ajavan roskienkeruun halutulla ajoituksella. Mittaustuloksien mukaan lisätään ote Androidin järjestelmälokista, mistä voidaan havaita mahdolliset häiriöitä aiheuttavat roskienkeruutapahtumat.

Kuva 5.9: Vasteajan muuttuminen suorituksen jatkuessa



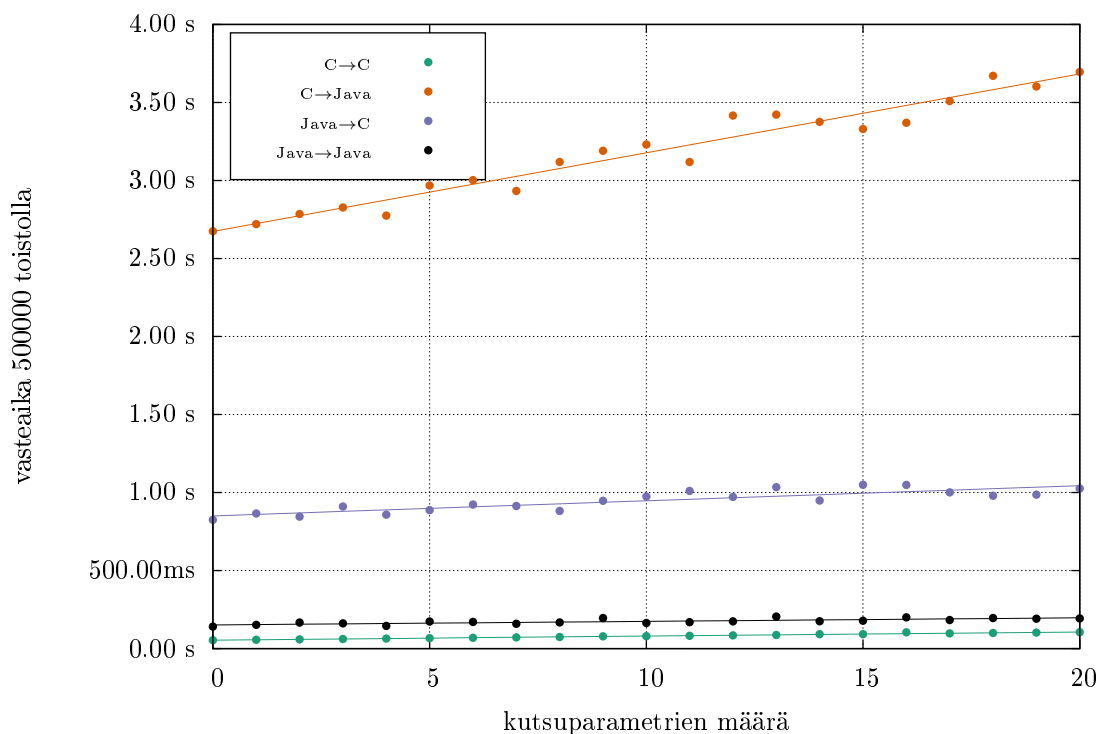
Viimeinen ongelma allokoivissa operaatioissa on, että käynnistettäessä Android-sovellukselle annetaan sallittua maksimikokoa pienempi keko, jonka kasvattaminen tarpeen mukaan taas saattaa aiheuttaa suorituskykyhäiriöitä. Ongelmaa on vältetty allokoimalla heti sovelluksen käynnistyessä suuri taulukko-olio, johon saatu viite tuhoetaan heti roskienkeruun mahdollistamiseksi.

JIT-käännös

Dalvik sisältää JIT-kääntäjän, jonka toiminta tekee Java-koodin käyttäytymisestä vaikeasti ennustettavaa: koska mittauksessa suoritetaan samaa silmukkaa jatkuvasti, on melko varmaa että se JIT-käännetään jossain vaiheessa, mutta suorituksen aluksi silmukan tavukoodia tulkitaan. Ongelma on ratkaistu suorittamalla ennen varsinaisia mittauksia *lämmittelyajo*, jonka kierrosmäärä on arvioitu riittäväksi JIT-kääntämisen laukaisemiseksi.

Oikean kierrosmäärän arvioimiseksi ja JIT-käännöksen havaitsemiseksi tehtiin mitaus, jossa samaa testiä suoritetaan pienehköllä silmukkakierrosluvulla mutta monessa perättäisessä ajossa, ja seurataan peräkkäisten vasteaikojen kehittymistä. Mittaustulokset näkyvät kuvassa 5.9. Vaaka-akselilla näkyy mittauskertta ja pystyakselilla vasteaika. JIT-käännös tapahtuu noin kierroksen 4700 kohdalla, minkä jälkeen vasteajat ovat systemaattisesti hieman pienempiä.

Kuva 5.10: int-tyyppiset kutsuparametrit



5.2 Mittaustulokset

Tässä luvussa esitellään kerätyt mittaustulokset ja koostetaan niistä alustavat havainnot. Myös mittauksen ulkopuolelle jätetyt operaatiot mainitaan perusteluineen.

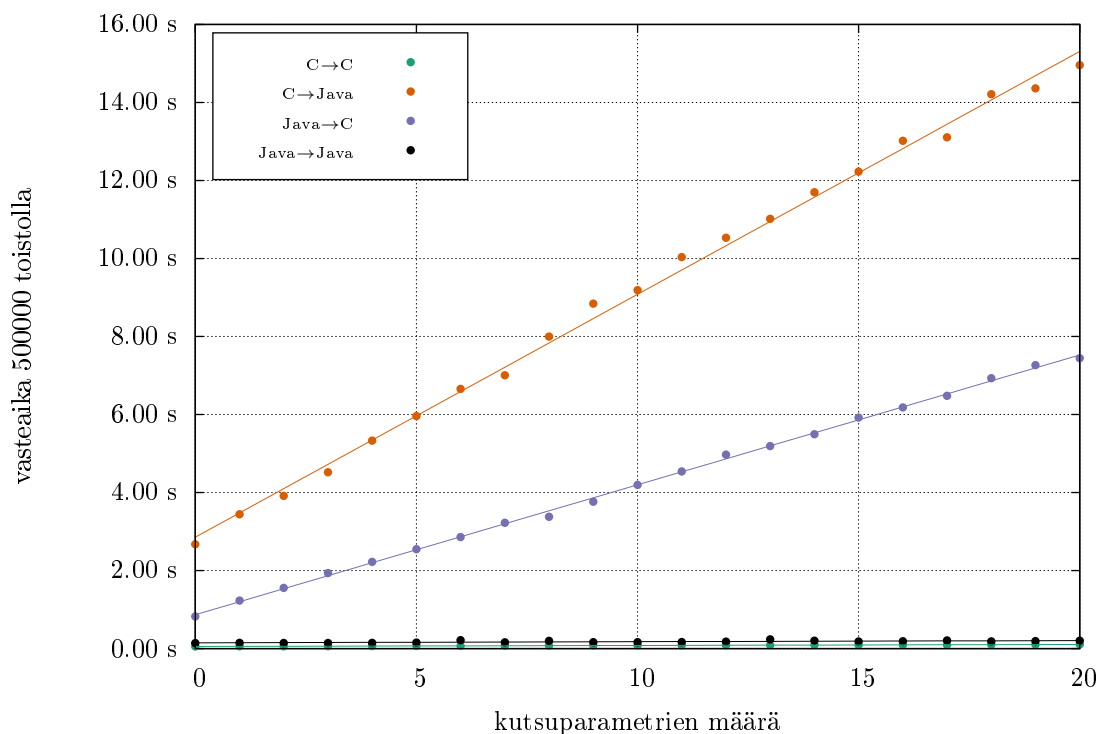
Metodien ja funktioiden kutsuminen

Mittauksen kannalta keskiössä ovat JNI-natiivirajapinnan kautta tehtävät aliohjelmakutsut erityyppisillä parametreilla. Operaatiot suunnassa C→Java ovat

1. `CallMethod`,
2. `CallNonVirtualMethod` ja
3. `CallStaticMethod`.

Operaatioista (1) huomioi olion luokkahierarkian eli kutsu sidotaan oikeaan luokkaan dynaamisesti kuten Javan metodikutsussa. Sen sijaan (2) kutsuu aina eksplisiittisesti määritellyn luokan metodia. (3) kutsuu staattista metodia. Variaatioita (1)-(3) vertaillaan mittauksissa.

Kuva 5.11: Object-tyyppiset kutsuparametrit



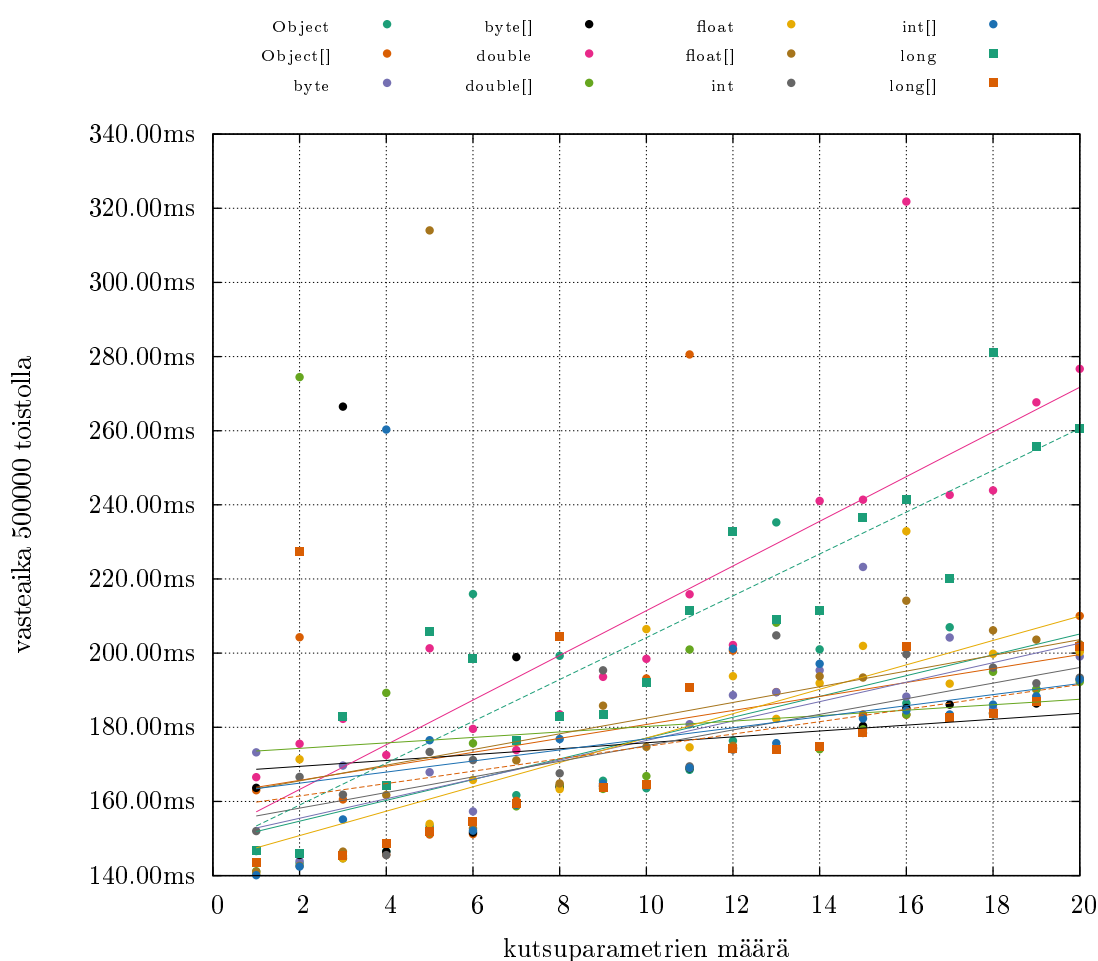
Metodien kutsuminen edellyttää aina metoditunnuksen noutamista `GetMethodID`- ja `GetStaticMethodID`-kutsuilla. Kuten suositeltu, palveluita ei mittauksissa kutsuta jokaisen metodikutsun yhteydessä vaan kerran koko mittausta käynnistettäessä.

Vastinpari suunnassa `Java→C` on `native`-metodin kutsuminen Javasta eli C-ohjelmassa olevan funktion kutsuminen. Variaatioille (1)–(2) ei ole vastinetta C-kielessä, mutta `native`-metodin voi merkitä staattiseksi (3). Vastinpari suunnassa `Java→Java` on tavallinen metodikutsu. Vastinpari on tavallinen funktiokutsu.

Kaikissa tuloksissa ja kaikissa permutaatioissa *vasteaika on lineaarisesti riippuvainen parametrien määrästä*. Kuitenkin *viitetyyppisillä parametreilla riippuvuuden kulmakerroin on suurempi*, eli viitetyyppisen parametrin käsittely näkyy tuloksissa keskeisenä suorituskykyrasitteena. Tätä havainnollistavat kuvat 5.10 ja 5.11, joissa parametrityypinä on `int` ja `Object`.

Kuvissa näkyvät kaikki kutsusuuntien permutaatiot, joten niistä voi myös hahmottaa eri kutsusuuntien vasteaikojen suhteellisia eroja. Kuten kohdassa 4.3 arvioitiin, Dalvikissäkin *kutsut suunnassa `Java→C` ovat selvästi päinvastaista suuntaa nopeampia*: edellisten vasteajat suhteessa jälkimmäisten aikoihin ovat noin kolminkertaisia. Tämä pätee kaikilla parametrityypeillä.

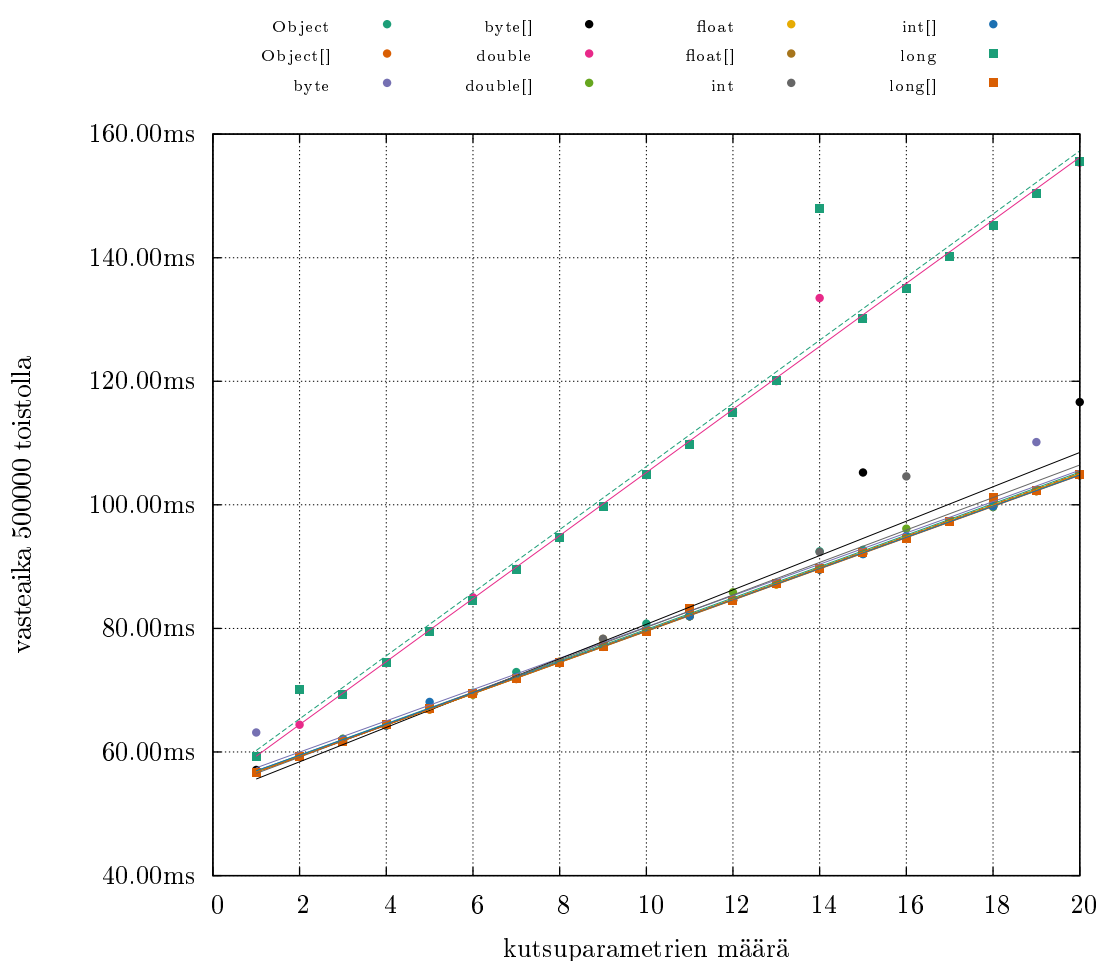
Kuva 5.12: Parametrityyppien vertailu Java→Java



Sen sijaan samankielisten suuntien (Java→Java) sekä (C→C) välisestä erosta ei tämän mittauksen perusteella voi tehdä pitkälle meneviä johtopäätöksiä muun muassa siksi, että mittausteknisistä syistä Java-lähtösilmukoihin on jouduttu lisäämään sivuvaikutuksellisia operaatioita, joita ei C-kielisissä silmukoissa tarvita, koska C-kääntäjästä on kytketty tarpeettoman koodin optimointi pois päältä. Näissäkin kutsusuunnissa vasteaika on kuitenkin lineaarisesti riippuvainen parametrimäärästä, vaikka näin pienillä vasteajoilla mittaushäiriöt ovat jo suhteellisesti merkittäviä. Riippuvuus näkyy kuvissa 5.12 ja 5.13, joihin on koostettu kaikki mittausten parametrityypit samankielisissä kutsusuunnissa. Lineaarisen riippuvuuden perusteella Java-kääntäjä tai Dalvikin JIT-kääntäjä ei ole onnistunut optimoimaan pois tyhjän metodin kutsua Java-koodista.

Kuvassa 5.13 mainitut Java-tyypit kuvautuvat C-kielen tyypeiksi seuraavasti. Viitetyypiset `Object` sekä taulukkotyypit kuvautuvat `jobject`-tyyppinimen kautta

Kuva 5.13: Parametrityyppien vertailu C→C

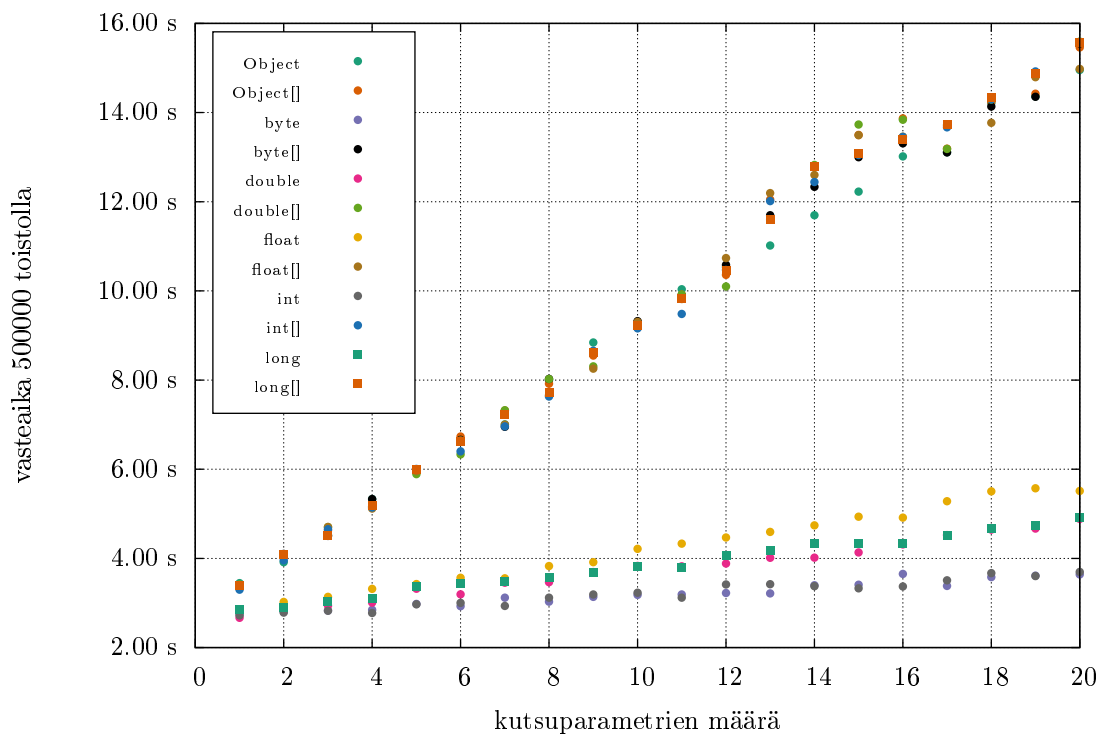


`void`-osoittimiksi ja primitiivityypit kuvautuvat taulukon 4.1 mukaisesti C-kielen perustyypeiksi.

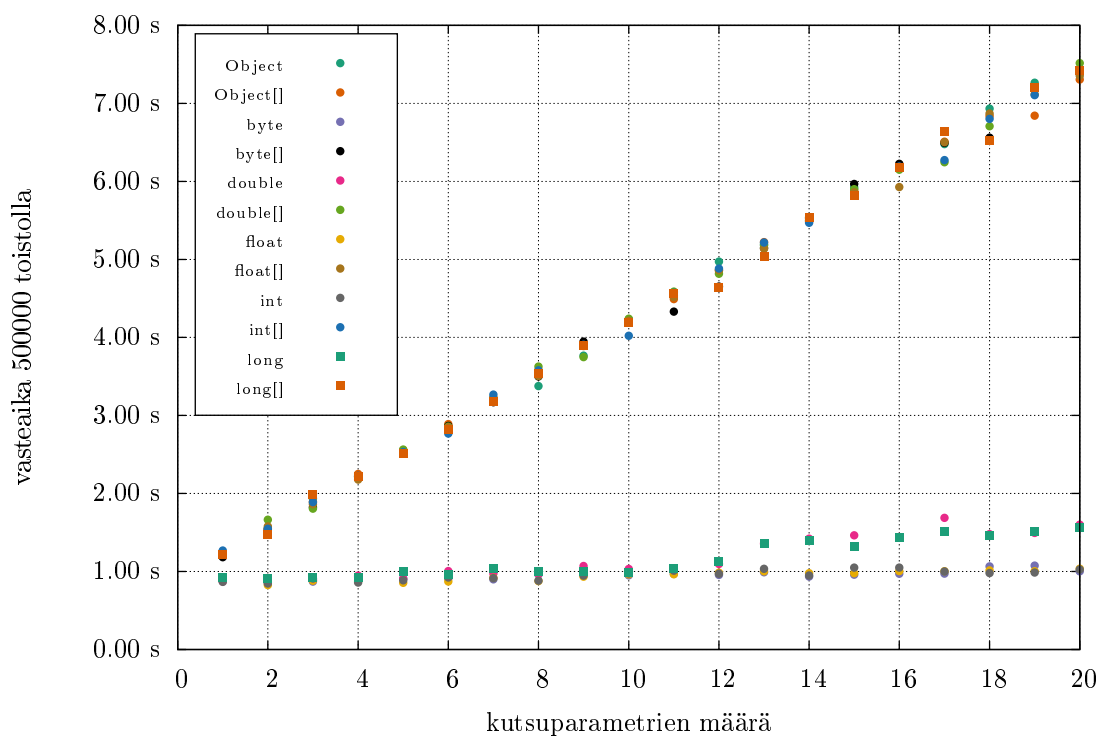
Kuviin 5.14 ja 5.15 on koostettu eri parametrityypit suunnissa (C→Java) ja (Java→C). Primitiivi- ja viitetyyppien ryhmittymisen näkyy kuvissa selvästi. Samankielisissä permutaatioissa (kuvat 5.12 ja 5.13) ei vastaavaa eroa näy, mutta mielenkiintoisena yksityiskohtana niissä voidaan havaita 64-bittisten parametrityyppien `long` ja `double` suurempi rasite 32-bittisessä prosessorissa.

Vasteajat riippuvat siis lineaarisesti parametrien määrästä, mutta eivät esimerkiksi parametrina välitettävän taulukon koosta (kuva 5.16), mikä olisikin odottamatonta koska taulukot välitetään viiteparametrina.

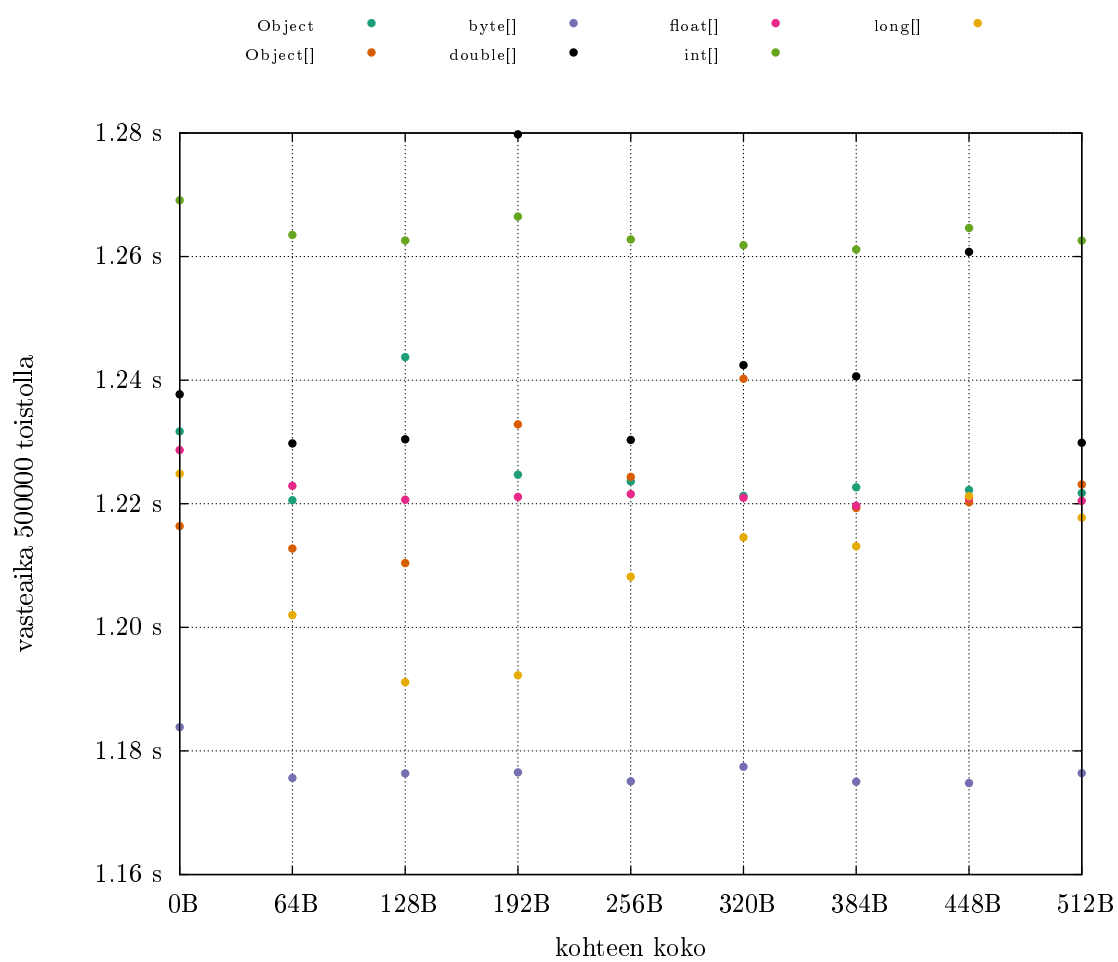
Kuva 5.14: Parametrityyppien vertailu C→Java



Kuva 5.15: Parametrityyppien vertailu Java→C



Kuva 5.16: Vaihteleva argumentin koko kutsusuunnassa Java→C



Kutsuprofiilit suunnassa C→Java

Kutsuprofiilien avulla saadaan lisävalaistusta viiteparametrien aiheuttamaan suurempaan rasitteeseen sekä kieltenvälisten kutsusuuntien keskinäisiin eroihin. Listauksen 5.1 kutsuprofilissa nähdään, missä aliohjelmissa suoritusaikaa kuluu eniten mittauksessa, jossa natiivikomponentista kutsutaan Java-metodia kahdellakymmenellä Object-tyyppisellä parametrilla. Suurin osa profileissa esiintyvistä aliohjelmista kuuluu Dalvikin C++-toteutukseen.

Tutkielman kutsuprofiilit on laskettu luvussa 3.3 sivulla 17 esitellyn inklusiivisen suoritusajan avulla, eli aliohjelman suoritusaikaa kuvaavissa prosenttiluvuissa on mukana aliohjelmassa itsessään sekä kaikissa sen välittömästi tai välillisesti kutsuissa aliohjelmissa vietetty aika.

Luettaessa profiilia ylhäältä alaspäin päästään kutsuvasta aliohjelmasta kutsuttuun aliohjelmaan. Profilissa näkyvä aliohjelma *A* on suorituksen aikana kutsunut suoraan kaikkia aliohjelmia, joiden prosenttiluku ja nimi on piirretty välittömästi aliohjelman *A* prosenttiluvusta lähtevän pystyviivan oikealle puolelle. Esimerkiksi `dvmDecodeIndirectRef` ❶ on kutsunut suoraan C++-luokan `ScopedPthreadMutex Lock` konstruktoria ❷ ja destruktoria ❸ sekä luokan `IndirectRefTable` metodia `get` ❹. Jos profilissa esiintyy saman prosenttiluvun alla useampi aliohjelma, nämä esiintyvät kutsupinossa aina yhdessä – esimerkiksi ketju nopeasti suoritettavia pieniä aliohjelmia, joita kutsuprofiilinäytteen ajallinen erottelukyky tai käytettyjen prosenttilukujen kahden desimaalin tarkkuus ei riitä erottamaan toisistaan.

Listauksen 5.1 kutsuprofilista ja muista tutkielman listauksista on jätetty pois perfmittauksista ja Androidin taustaprosesseista aiheutuva rasite. Listauksessa 5.1 tämä rasite on noin 28%, minkä takia profiilin kokonaissuoritusaika ei ole 100%. Listaus 5.1 on rajattu alkamaan natiivikirjaston `runInternal`-natiivimetodista.

Mitattavassa metodissa on keinotekoisien paljon viiteparametreja, joten profilista erottuu viitteiden välittämisen aikarasite. Avainkohta on Dalvikin metodi `dvmDecodeIndirectRef`, joka vie 46.21% koko suoritusajasta ja 64% testin suoritusajasta. *Indirect reference* tarkoittaa epäsuoraa viitettä, joka välitetään parametrinä rajapinnan yli: JNI-kutsujen `jobject*`-tyyppisiä parametreja eri muodoissaan. Dalvik-virtuaalikone pitää kirjaa välitetyistä viitteistä tarkoitusta varten toteutetulla taulutietorakenteella (`dalvik/vm/IndirectRefTable`). Epäsuoria viitteitä tarvitaan, jotta virtuaalikoneen muistinhallinnalla säilyy tieto myös JNI:n yli välitetyistä viitteistä. Toisaalta natiivikomponentti ei voi käsitellä JNI-operaatioilla viitetyyp-

Listaus 5.1: Metodikutsu C→Java 20 viiteparametrilla



pisiä olioita suorilla muistiosoitteilla senkään vuoksi, että virtuaalikoneella säilyisi mahdollisuus järjestellä hallitsemiaan muistialueita uudelleen.

Profilissa yli puolet epäsuorien viitteiden hallinnan suoritusajasta kuluu rinnakkaisen säikeiden poissulkemiseen `ScopedPthreadMutexLock`- olion avulla. Tämä johtuu siitä, että mittauksessa välitetyt olioviitteet ovat *globaaleja* viitteitä, joita säilytetään Dalvikin kaikille säikeille yhteisessä, globaalissa viitetaulukossa. Mittauksessa käytetään globaaleja viitteitä lähinnä ohjelmakoodin yksinkertaistamiseksi, minkä seurauksena löytyikin sattumalta tämä hitain suorituspolku `dvmDecodeIndirectRef` -metodin sisältä. Listaus 5.2 on ote Dalvik-virtuaalikoneen lähdekoodista. Siinä näkyy mutex-lukon varaaminen kohdassa ❶. Nopeampi suorituspolku käyttää lokaalin JNI-kontekstin omaa viitetaulukkoa ilman lukitusta ❷

Listaus 5.2: Ote funktiosta `dvmDecodeIndirectRef`

```
switch (indirectRefKind(jobj)) {
case kIndirectKindLocal:
{
    Object* result = self->jniLocalRefTable.get(jobj); ❷
    if (UNLIKELY(result == NULL)) {
        ALOGE("JNI_ERROR(app_bug):_ "
              "use_of_deleted_local_reference(%p)", jobj);
        dvmAbort();
    }
    return result;
}
case kIndirectKindGlobal:
{
    // TODO: find a way to avoid the mutex activity here
    IndirectRefTable* pRefTable = &gDvm.jniGlobalRefTable;
    ScopedPthreadMutexLock lock(&gDvm.jniGlobalRefLock); ❶
    Object* result = pRefTable->get(jobj);
    if (UNLIKELY(result == NULL)) {
        ALOGE("JNI_ERROR(app_bug):_ "
              "use_of_deleted_global_reference(%p)", jobj);
        dvmAbort();
    }
    return result;
}
}
```

Kattavammissa mittauksissa täytyisi laatia erilliset testit, jotka käyttävät paikallisia viitteitä. Tämänkin mittauksen avulla saadaan kuitenkin jotain käsitystä nopeutuksen suuruusluokasta, joka paikallisilla viitteillä saavutettaisiin: tämän testin kokonaissuoritusajasta yli kolmasosa kuluu lukitukseen.

Listauksen 5.3 kutsuprofilissa kutsusuunta on $C \rightarrow \text{Java}$ ja kutsussa on kaksikymmentä `int`-parametria. Profilissa näkyy edelleen `dvmDecodeIndirectRef` (❶)

Listaus 5.3: Metodikutsu $C \rightarrow \text{Java}$ 20 int-parametrilla

13.42% suoritusajalla — testissä kutsutaan Java-olion kohdemetodia, joten olio täyttyä välittää viitteenä JNI-kutsussa.

```
(*env)->CallVoidMethod(env, java_counterparts_object, mid);
```

Kutsussa välitetään siis viite `java_counterparts_object` -olioon, jonka metodia ollaan kutsumassa. Viitteidenkäsittely ei kuitenkaan enää dominoi kutsuprofilia, vaan merkittäviä ovat myös muut $C \rightarrow Java$ -kutsun perusrasitteet, jotka ovat nähtävissä profiilin haaran `CallVoidMethod` alakohdissa.

`dvmCallMethodV` (15.56%) vastaa Dalvikissa metodin kutsumisesta vaihtelevalla argumenttimäärällä kaikissa kutsusuunnissa. Ainoa JNI:hin liittyvä rasite siitä alkavassa kutsuhaarassa on `callPrep`-metodi, jota ei näy vastaavassa vertailukohdassa suunnassa $Java \rightarrow Java$. Suurimpana yleisrasitteena kyseisessä haarassa näkyvät JIT-kääntäjän suorituksen jäljittämiseen (tracing) liittyvät toimet, jotka näkyvät myös suunnassa $Java \rightarrow Java$.

Varsinaiset JNI-rajapinnan käyttöön liittyvät rasitteet näkyvät kutsussa `dvmGetVirtualizedMethod` (3.35%), joka vastaa dynaamisesti sidotun metodin löytämisestä, sekä `ScopedJniThreadState`-olion luomisesta ja poistamisesta (yhteensä 5.52%). Olion kautta muutetaan virtuaalikoneen sisäinen tila `dvmChangeStatus`-metodilla. Tämä tilanmuutos tapahtuu kaikissa JNI-kutsuissa, ja sen avulla virtuaalikone pitää kirjaa, milloin muistinhallinnan roskienkeruu on sallittua.

Säikeen suorituksen ollessa natiivikomponentissa tila on `THREAD_NATIVE`, jolloin roskienkeruu on mahdollista, mutta Dalvikin viitetaulukkoa lukemalla roskienkeruu pysyy tietoisena JNI-rajapinnan yli välitetyistä olioviitteistä. Suorituksen ollessa käynnissä Java-komponentissa tila on `THREAD_RUNNING`, jolloin roskienkeruu edellyttää tilamuutosta esimerkiksi `THREAD_SUSPENDED`-tilaan (johon tilaan virtuaalikone asettaa itsensä ennen roskien keruuta).

Staattiset metodikutsut suunnassa $C \rightarrow Java$

Viitteidenkäsittelyn lisärasitteesta Dalvikissa voi päästä eroon käyttämällä pelkästään primitiiviparametreja ja kutsumalla Java-luokan *staattista metodia*.

Profilissa 5.4 kutsuttu metodi on staattinen, eikä `dvmDecodeIndirectRef`-kutsua näy. Kun staattista metodia kutsutaan, välitetään JNI-rajapinnan yli `jclass`-viite, mutta Dalvikissa ei tällaisesta luokkaviitteestä siis pidetä kirjaa viitetaulukon avulla. Rasite on 500 000 kierroksella 1.49 sekuntia, kun vastaavalla dynaamisella kutsulla

Listaus 5.4: Staattinen metodi C→Java

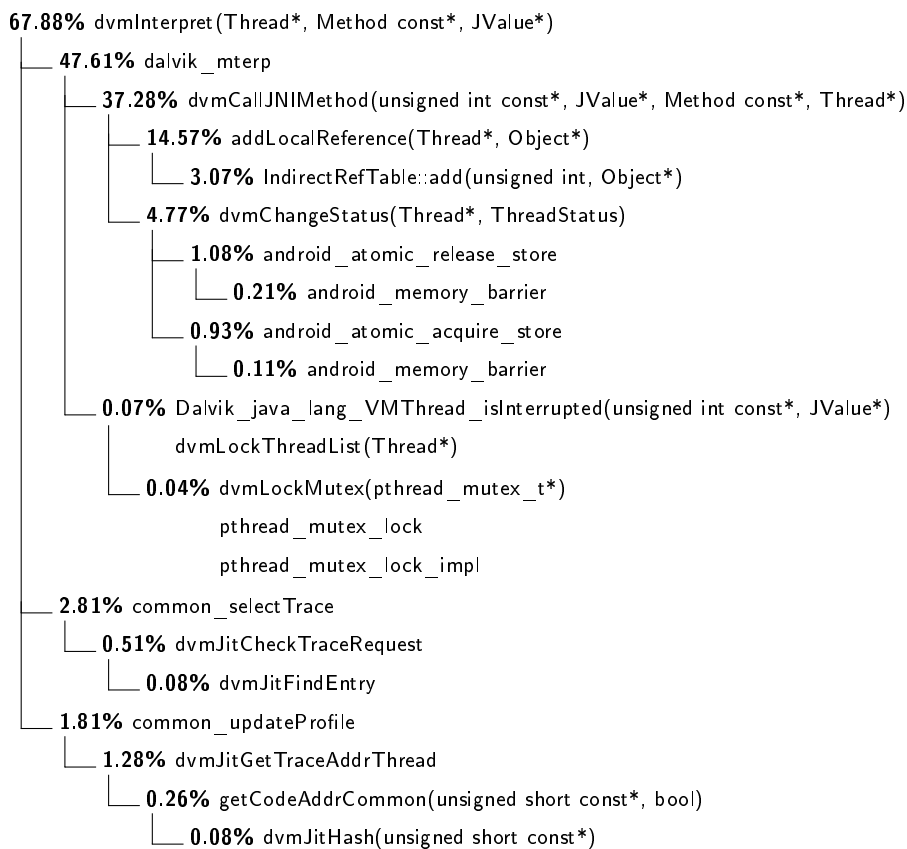


se on 2.68 sekuntia, mistä suurin osa selittyy juuri kutsujen `dvmDecodeIndirectRef` sekä `dvmGetVirtualizedMethod` poistumisella. Mittauksissa ei ole suoritettu kattavasti staattisia metodikutsuja vaihtelevilla parametrimäärillä, vaan ainoastaan parametrin määrällä 0, jotta eri kutsutyyppeiden variaatioiden määrä pysyisi hallinnassa.

Kutsuprofiilit suunnassa Java→C

Mikä sitten selittää, että C→Java -kutsusuunnan vasteaika on 1.56 kertaa Java→C-kutsusuunnan vasteaika parhaimmassakin tapauksessa eli kutsuttaessa staattista metodia ilman parametreja? Luvussa täytyy huomioida, että mitatut kokonaisvas-

Listaus 5.5: Staattinen metodi Java→C

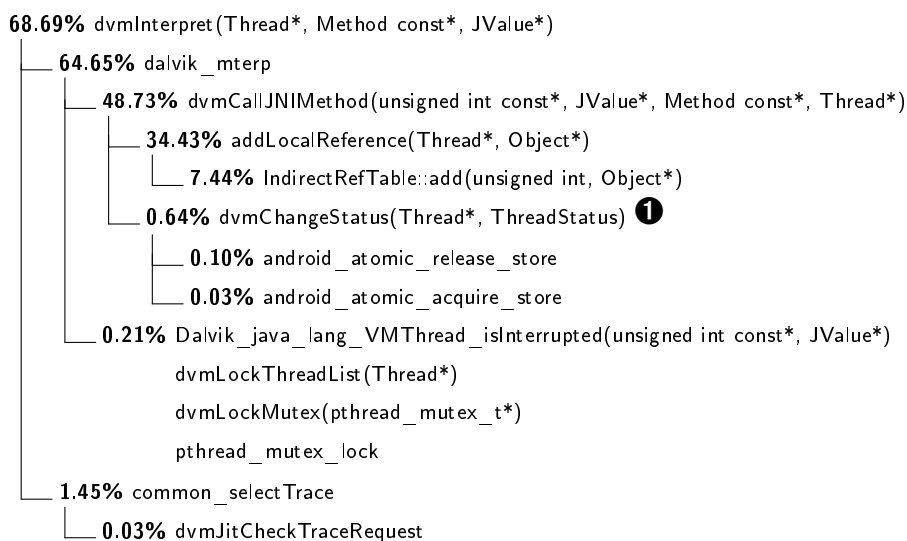


teajat sisältävät muutakin kuin tarkastelun kohteena olevassa JNI-toteutuksessa vietetyn ajan — nimittäin kutsun kohteena olevan aliohjelman suorittamiseen kuluvan ajan.

Java-komponentissa olevan kohdemetodin suoritus konekielitasolla on hyvin erilais- ta natiivikomponentin kohdefunktion suoritukseen nähden. Edellistä suoritetaan Dalvikin tavukooditulkkin kautta, joka huolehtii myös JIT-kääntämiseen liittyväs- tä jäljittämisestä ja suorittaa mahdollisesti JIT-käännettyä koodia. Lisäksi Java- kohdemetodiin on näissä mittauksissa lisätty yksinkertainen sivuvaikutuksellinen operaatio, jotta sitä ei optimoitaisi tyhjänä pois käännösvaiheessa.

JNI-lisärasitteen suuruudesta saakin paremman arvion seuraavasti. Kutsusuun- nan C→Java rasite saadaan vähentämällä (C→Java)-vasteajasta (Java→Java)- vasteaika. Sama pätee kääntäen myös toiseen kutsusuuntaan. Olettaen, että itse läh- tösilmmukan yleisrasite on merkityksettömän pieni suhteessa koko vasteaikaan, saa-

Listaus 5.6: 20 Viiteparametria Java→C

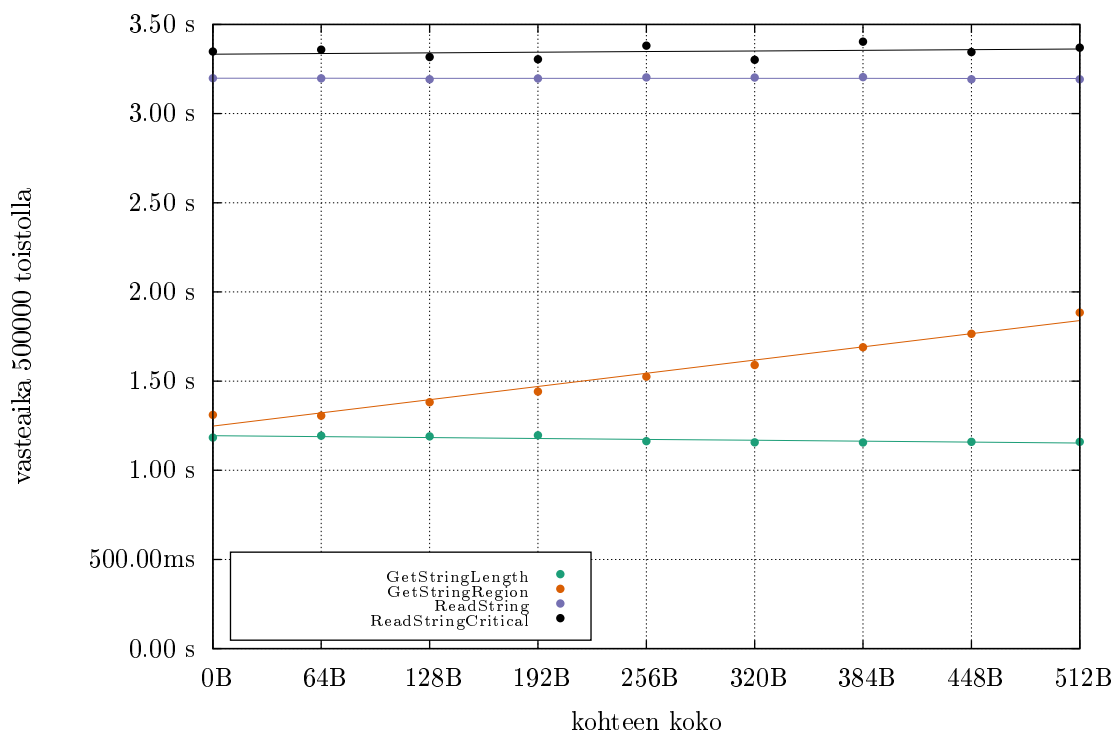


daan näin selville, miten paljon kauemmin kestää kutsua samaa kohdealiohjelmaa JNI:n yli verrattuna normaaliin kutsuun ilman JNI:tä. Tällä tavalla saadaan tarkennettua C→Java kutsusuunnan JNI-rasite tässä parhaimmassa tapauksessa noin 1.19 kertaiseksi Java→C-suunnan rasitteeseen nähden.

Listauksien 5.4 ja 5.5 profileja vertailemalla saadaan tähän lisärasitteeseen ainakin osittainen selitys. Listauksessa 5.4 on staattisen metodikutsun profili suunnassa C→Java ja listauksessa 5.5 on vastaava kutsuprofili suunnassa Java→C. Java→C-kutsussa ei luoda `ScopedJniThreadState`-oliota, vaan virtuaalikoneen suoritustilan muuttavaa `dvmChangeStatus`-metodia kutsutaan suoraan. Samoin `callPrep`-funktio puuttuu. Näiden toteutusyksityiskohtien tarkempi analyysi jätetään tässä tekemättä.

Kuvasta 5.11 nähdään myös, että viiteparametrien määrän lisääntyessä Java→C-suunnan kuvaajan kulmakerroin on loivempi. Tästä voisi päätellä, että viitteiden hallinta tähän suuntaan on tehokkaampaa. Listauksen 5.6 perusteella ero saattaa suurelta osin selittyä sillä, että Java→C-suunnassa käsitellyt viitteet ovat lokaaleja ja päinvastaisen suunnan viitteet globaaleja. Kyseisessä suunnassa viitteet ovat oletusarvoisesti lokaaleja JNI-standardin mukaisesti – natiivikomponentin on erikseen luotava globaalit viitteet, jos aikoo käyttää viitteitä yksittäisen natiivimetodikut-

Kuva 5.17: Unicode-merkkijonot suunnassa C→Java



sun elinkaaren ulkopuolella. Suunnassa Java→C ei myöskään tarvita rinnakkaisten säikeiden poissulkemista listauksen 5.6 kohdassa ❶.

Merkkijonojen käsittely

Java-merkkijonojen sisältöä käsitellään natiivikomponentista seuraavilla operaatioilla. Osoittimen palauttavia ja vapauttavia operaatioita kutsutaan aina pareittain, joten jokainen pari on koottu yhteisen testin alle. On tärkeä huomata, että Java-merkkijonojen muokkaaminen on ohjelmoijalta kielletty, vaikka se olisikin mahdollista JNI:n osoitinoperaatioiden avulla: Javan merkkijonot ovat muuttumattomia. Mittauksissa testeille on annettu nimet seuraavasti:

ReadString **GetStringChars** ja **ReleaseStringChars**

ReadStringCritical **GetStringCritical** ja **ReleaseStringCritical**

ReadStringUTF **GetStringUTFChars** ja **ReleaseStringUTFChars**.

Kopioivat operaatiot **GetStringRegion** ja **GetStringUTFRegion** näkyvät tuloksissa nimillään samoin kuin tukiopeatiot **GetStringLength** ja **GetStringUTFLength**.

Viimeksi mainituilla selvitetään merkkijonon pituus, mikä on tarpeen ennen kuin merkkijonon voi läpikäydä C-koodissa.

Näille $C \rightarrow \text{Java}$ -suunnan operaatioille ei ole löydettävissä vastinpareja suunnissa $C \rightarrow C$ tai $\text{Java} \rightarrow \text{Java}$, sillä kyseessä ovat operaatiot, joita on kutsuttava sen *lisäksi* että merkkijonoa varsinaisesti luetaan natiivikoodissa. Siten natiivimerkkijonon käsittely C-kielessä ei vaadi mitään vastaavaa ylimääräistä operaatiota kuten ei myöskään Java-merkkijonon käsittely Java-koodissa, vaan kyseessä on puhtaasti JNI-rajapinnasta johtuva lisärasite.

Kutsusuunnassa $\text{Java} \rightarrow C$ operaatioille on kuitenkin eräs vertailukohta. Natiivimerkkijonon eli `char`-taulukoiden käsittely `java.nio.CharBuffer`-olioina Java-koodissa on mahdollista `Non-blocking IO-API:n` kautta, jota käsitellään myöhemmin tässä luvussa.

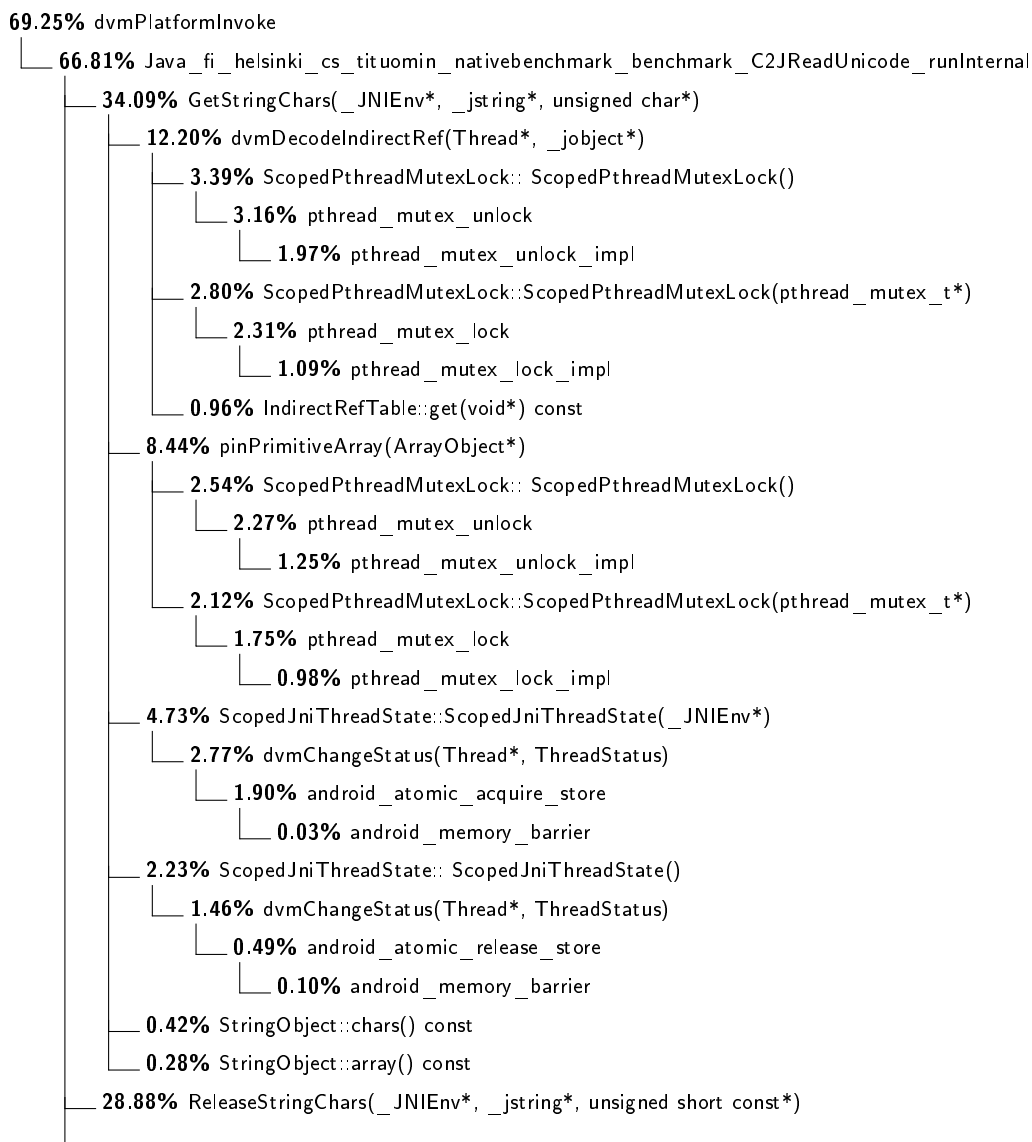
Vasteaikamittauksista huomataan ensinnäkin, että osoitinoperaatiot, jotka välittävät viittauksen suoraan virtuaalikoneen sisäiseen Unicode-tietorakenteeseen, ovat yhtä nopeita riippumatta merkkijonon pituudesta (ks. kuva 5.17). Tämä on odotusten mukaista, sillä operaatiossa välitetään vain osoite merkkijonoon. Myös Unicode-merkkijonon pituuden selvitys on vakiollinen operaatio, oletettavasti koska merkkijonon pituus on tallessa Java-merkkijonon sisäisessä toteutuksessa. Kopioivan operaation `GetStringRegion` riippuvuus on odotetusti lineaarinen.

Kuitenkin noin alle 1500 merkin pituisilla merkkijonoilla merkkijonon kopioiminen `GetStringRegion`-metodilla on nopeampaa kuin osoitteen saaminen merkkijonon muistialueelle. Osoitinoperaation kutsuprofiili näkyy listauksessa 5.7, jossa merkittävimpänä vasteajan komponenttina on jälleen globaalin `jstring`-viitteen purku. Seuraavaksi merkittävin kustannus (8.44%) syntyy toisesta muistinhallintaan liittyvästä toimesta, eli merkkijonon sisäisen merkkitaulukon *kiinnikyttemisestä* (`pinPrimitiveArray`), mikä takaa että palautettu muistiosoite pysyy voimassa. Itse merkkijonon merkkien nouto (`StringObject::chars()` ja `StringObject::array()`) vievät kokonaisajasta vain 0.7%. Muistinhallintaan liittyvät rasitteet tavallaan myös tuplaantuvat, koska vastaavat toimet tehdään sekä noudettaessa että vapautettaessa viite merkkijonon sisältöön, mikä näkyy myös kutsuprofilissa.

Kopioivan Unicode-operaation profilissa 5.8 näkyy myös globaalin `jstring`-viitteen purku, mutta se tapahtuu vain kerran eikä kahdesti.

Sen sijaan UTF-8-muunnoksen tekevä osoitinoperaatio on riippuvainen kohteen koosta. Se on myös operaatio, joka johtaa Dalvikin allokoimaan muistia muunnet-

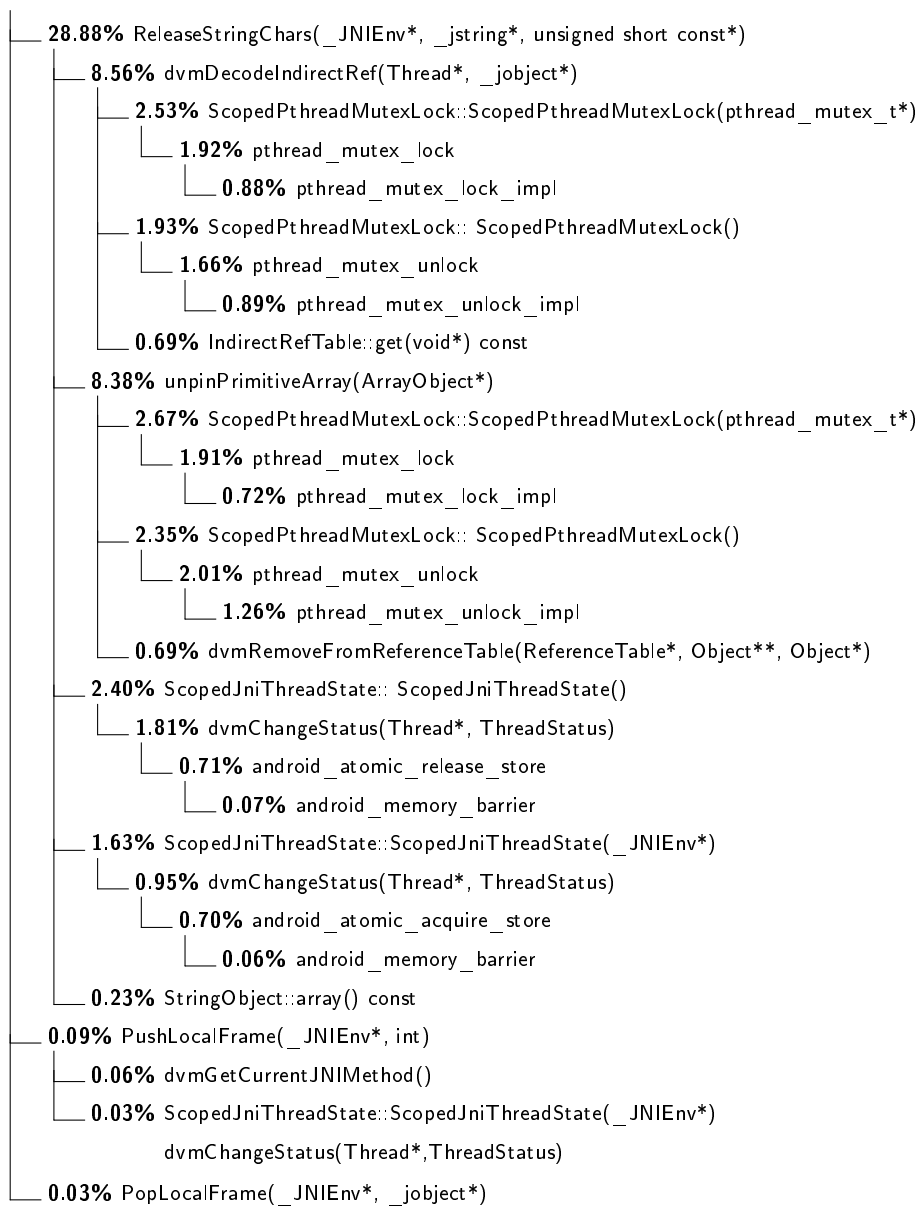
Listaus 5.7: Unicode-merkkijonon osoitinoperaatio, osa 1/2



tua merkkijonoa varten. Siksi kyseisen operaation vasteajat kuvassa 5.18 on mitattu allokoiiviin operaatioihin soveltuvilla vähäisemmällä lähtösilmukan kierrosmäärillä: allokoiivien operaatioiden kierrosluku on 300 ja ei-allokoivien 500000, joten vasteaikojen suhdeluku on 3:5000.

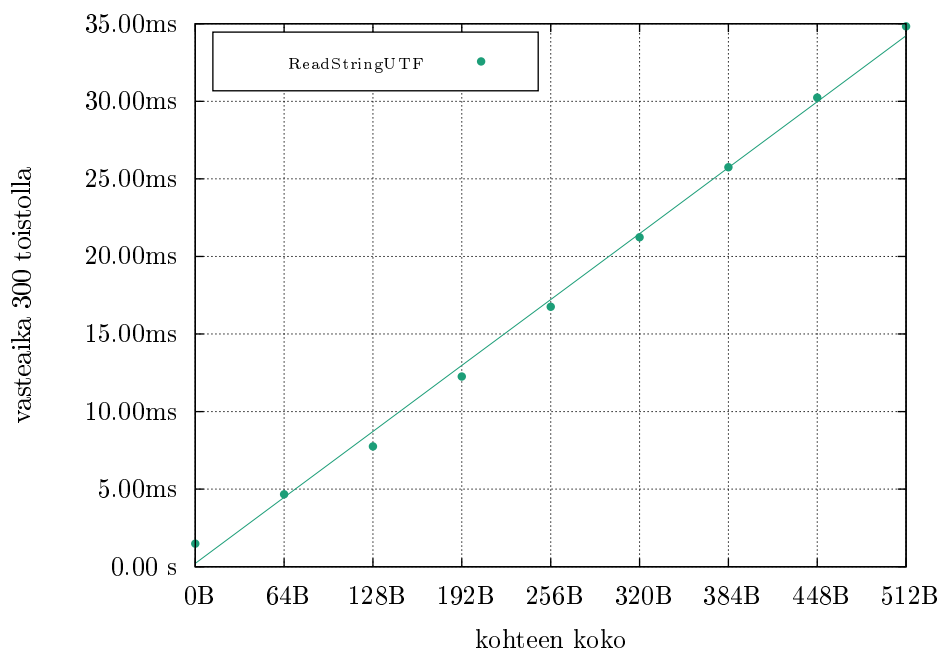
Samoin kopioivat operaatiot ovat lineaarisesti raskaampia kopioitavan kohteen koon kasvaessa, mutta tässäkin tapauksessa merkistökoodauksen muunnos Javan 16-bittisestä Unicodesta 8-bittiseen UTF-8:aan on merkittävästi raskaampi kuin pelkkä

Listaus 5.7: Unicode-merkkijonon osoitinoperaatio, osa 2/2

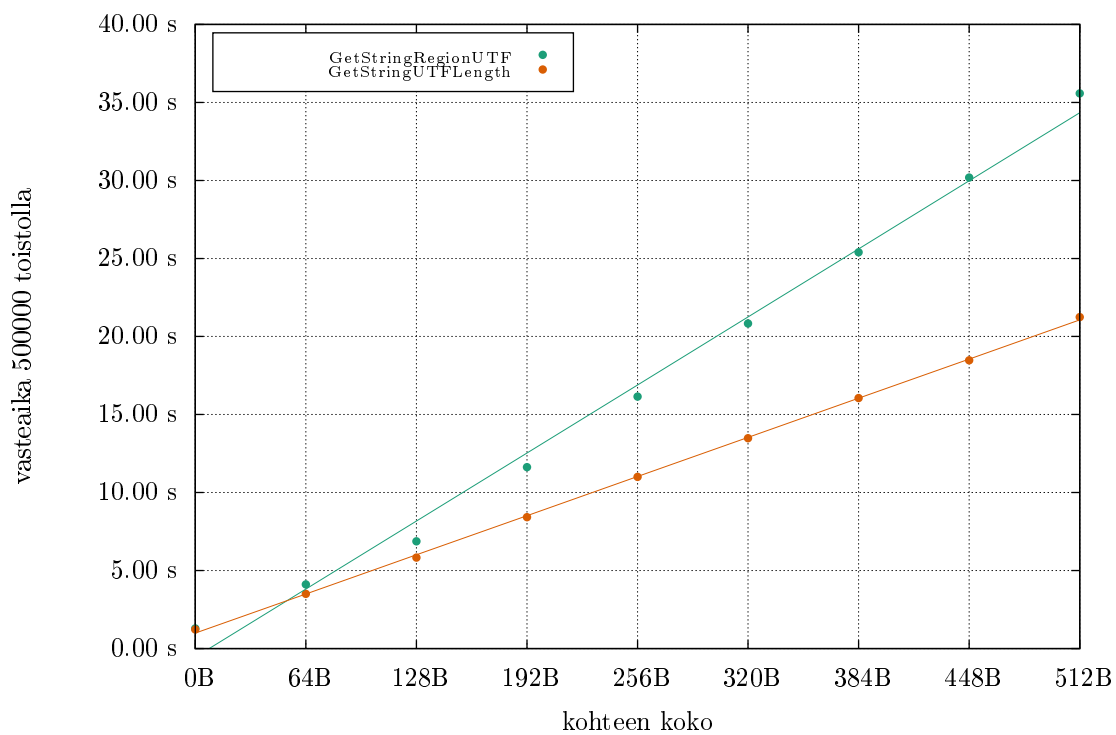


Unicode-merkkien kopiointi. Näin ollen kuvassa 5.19 merkkijonon koon selvittäminen UTF-8-koodauksessa on raskaampi operaatio kuin Unicode-merkkijonon kopiointi.

Kuva 5.18: UTF-merkkijonot suunnassa C→Java



Kuva 5.19: UTF-merkkijonot suunnassa C→Java



Listaus 5.8: Unicode-merkkijonon kopiointi C→Java



Taulukoiden käsittely

Taulukoiden käsittelyn perusmenetelmät ovat samat kuin merkkijonojen käsittelyssä, mutta merkistökoodaukseen liittyviä rasitteita ei tällöin ole, ja viitetyyppejä sisältävät taulukot käsitellään omana erityistapauksenaan. Lisäksi Java-tilukoiden lukeminen ja kirjoittaminen ovat molemmat sallittuja. Osoittimen palauttavia operaatiota mitataan seuraavissa testeissä.

GetArrayElements Funktiot `GetArrayElements` ja `ReleaseArrayElements`

ReadPrimitiveArrayCritical Funktiot `GetPrimitiveArrayCritical` ja `ReleasePrimitiveArrayCritical`

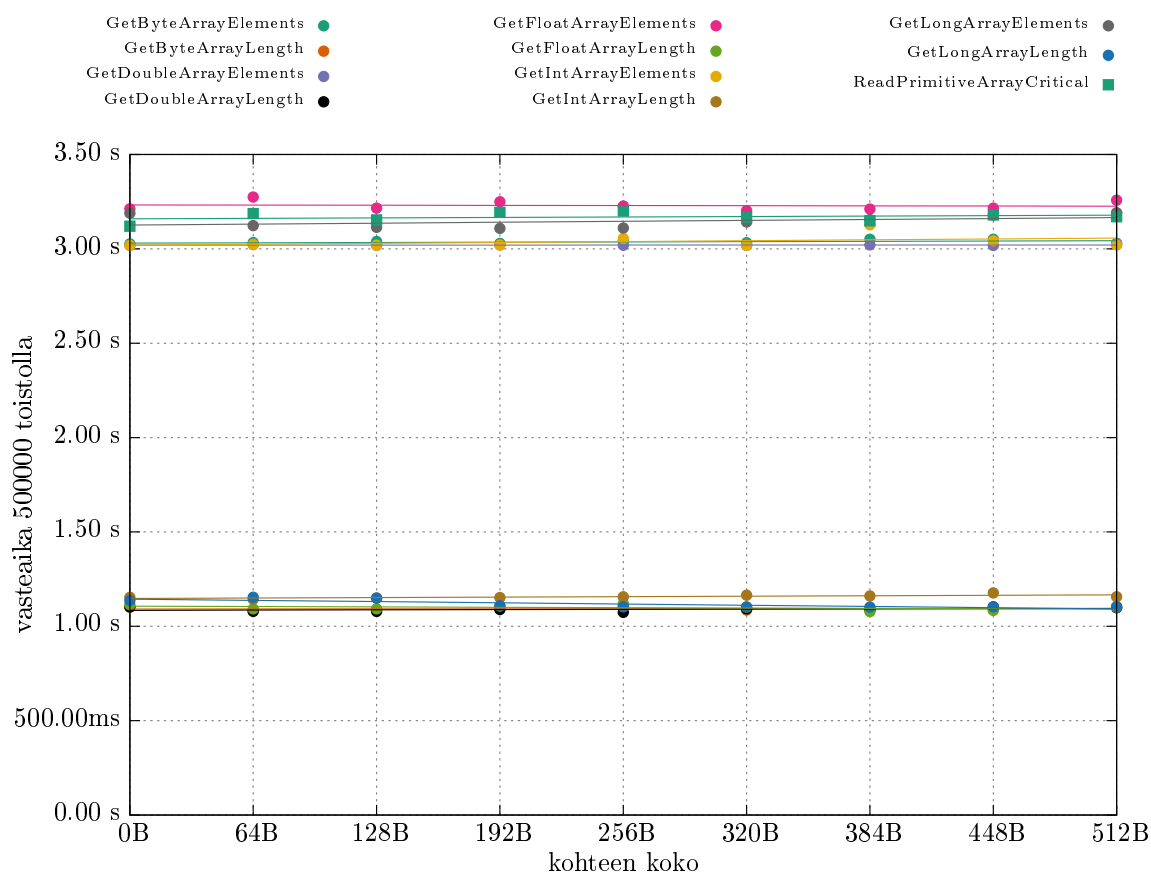
Operaatioiden nimet saadaan korvaamalla τ jollakin Javan primitiivityypeistä. Kopioivat operaatiot ovat `Get τ ArrayRegion` ja `Set τ ArrayRegion`.

Viitetyyppisten taulukoiden elementtejä voi käsitellä vain yksi kerrallaan operaatioilla `GetObjectArrayElement` ja `SetObjectArrayElement`. Molempien kutsurasitteeksi saatiin 46.6ms kutsua kohti eli karkeasti arvioiden yli 20 sekuntia 512-alkioisen taulukon läpikäyntiin.

Suoria vertailukohtia näille suunnan $C \rightarrow Java$ taulukko-operaatioille ei ole muissa kutsusuunnissa, jälleen kerran lukuun ottamatta NIO-rajapintaa (ks. kohta 5.2).

Primitiivitaulukoiden operaatioiden osoitinversioiden vasteajat ovat odotetusti vakiollisia suhteessa syötteen kokoon (kuvan 5.20 ylempi ryhmittymä), kun taas kopioivat versiot kasvavat lineaarisesti syötteen koon kasvaessa (kuva 5.21). Merkkijonon pituuden selvittäminen on vakiollinen operaatio (kuvan 5.20 alempi ryhmittymä).

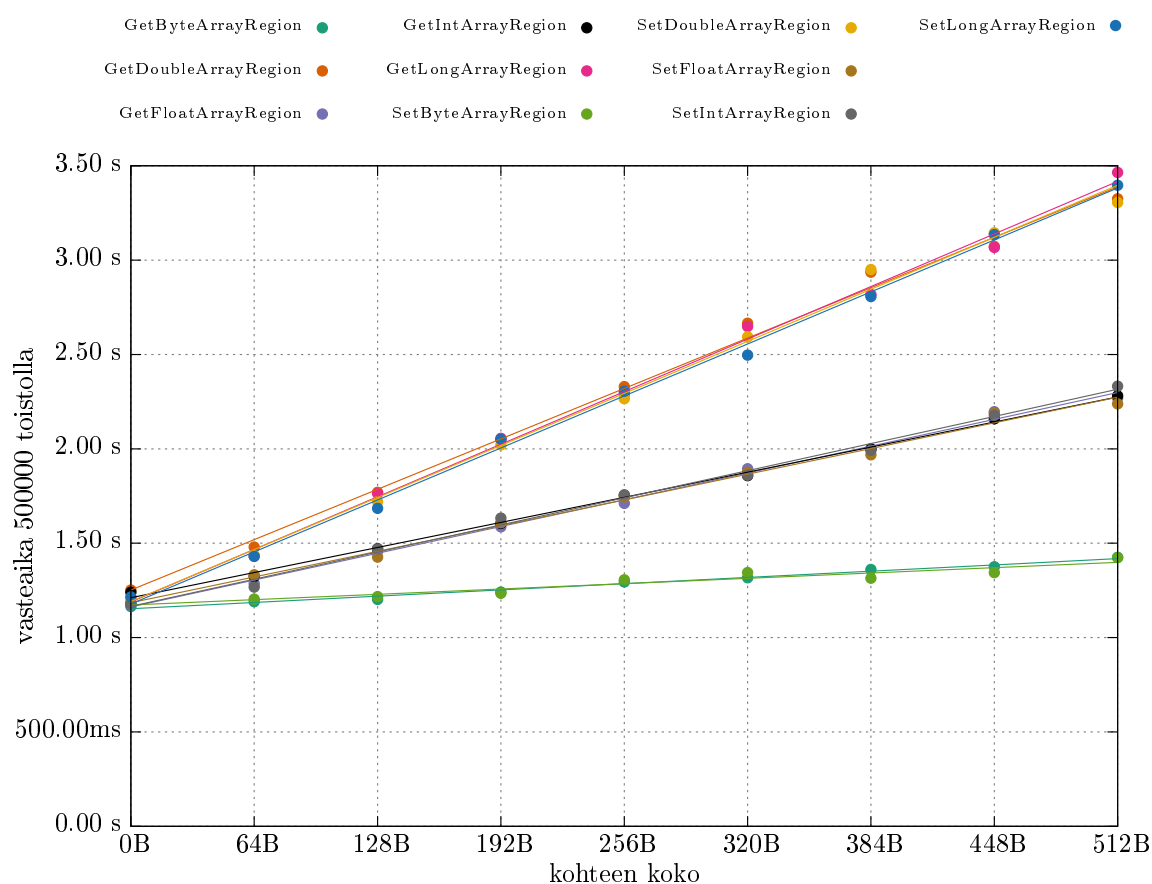
Kuva 5.20: Taulukoiden käsittely osoittimilla suunnassa $C \rightarrow Java$



Kutsuprofiilien perusteella taulukoiden osoitinoperaatioiden kutsurakenteet ovat täsmälleen samoja merkkijonojen osoitinoperaatioiden kanssa, joten kutsuprofiileja ei ole erikseen listattu. Funktioiden `GetCharArrayElements` ja `GetStringChars` vasteajatkin ovat mittauksissa yhtä suuret. Keskeisimmät rasitteet syntyvät siis epäsuoran globaalin viitteen purusta ja taulukon kiinnikytkemisestä.

Kopioivien operaatioiden vasteajat noudattelevat kopioitavan primitiivityypin kokoa. Vasteajoissa on vakioelementti, joka syntyy taulukkoviitteen purusta — kiinnikytkemistä ei kopioitaessa tarvita. Lineaarisen riippuvuuden kulmakerroin määräytyy melko suoraan varsinaisesta muistialueen kopioinnista (`memcpy`), määräävänä tekijänä kopioitavien elementtien koko tavuina. Siten `double`-taulukon kopiointi (kun vakiorasite on vähennetty) on noin kaksi kertaa niin raskasta kuin `float`-taulukon. Vertailun vuoksi kopioivan merkkijono-operaation `GetStringRegion` rasite vastaa hyvin 16-bittisen `char`-taulukon kopioimisen rasitetta. Kutsuprofiileissa lineaarinen

Kuva 5.21: Taulukoiden käsittely kopioimalla suunnassa $C \rightarrow \text{Java}$



riippuvuus näkyy niin, että `memcpy`-operaation prosenttiosuus operaatiossa kasvaa tasaisesti syötteen koon kasvaessa.

Non-blocking I/O

Javan standardikirjaston paketin `java.nio` luokka `ByteBuffer` antaa joitakin lisämahdollisuuksia datan siirtoon Java- ja natiivikomponenttien välillä. `ByteBuffer`-olio edustaa jatkuvaa muistialuetta, jota voi käsitellä tehokkaasti molemmissa ympäristöissä.

Jos `ByteBuffer`-on allokoitu *suorana* (*direct*), virtuaalikone yrittää taata, että natiiviooperaatiot (käyttöjärjestelmän operaatiot tai JNI-natiivikomponentin suorittamat operaatiot), joita puskurille suoritetaan, tehdään suoraan samalle muistialueelle, jota Java-ohjelmakin käsittelee. Sitä vastoin epäsuoran puskurin sisältö saatetaan kopioida erikseen ennen natiivi-I/O-operaatioita. JNI-rajapinta tarjoaa pääsyn ainoastaan *suoran* `ByteBuffer`-olion muistialueelle, jota voi käsitellä suoraan osoittimen avulla.

NIO-luokkien käytön kustannuksiin täytyy laskea seuraavien `ByteBuffer`-olion käsittelyssä välttämättömien JNI-palveluiden lisärasitteet. Ensimmäisessä vaihtoehdossa funktiolla `NewDirectByteBuffer` luodaan natiivikomponentista käsin suora tavupuskuri eli Javan `DirectByteBuffer`-olio *valmiiksi allokoituun muistialueeseen*.

Jos taas puskurin muisti on varattu Java-komponentissa, natiivikomponentti saa puskurin edustaman muistialueen osoitteen `GetDirectBufferAddress`-kutsulla ja kapasiteetin `GetDirectBufferCapacity`-kutsulla. Koska tässä vaihtoehdossa molemmat operaatiot ovat edellytyksiä puskurin käytölle, ne mitattiin yhdessä testissä, jonka vasteajaksi saatiin 2.31 sekuntia. Testin kutsuprofilissa näkyvät ainoastaan epäsuoran globaalin viitteen purku sekä virtuaalikoneen suoritustilan muutos, jotka nähtiin aiemminkin käsitellyissä JNI-kutsuissa.

JNI-kutsu	kierrosmäärä	vasteaika	vertailuluku
<code>GetDirectBufferAddress</code> + <code>GetDirectBufferCapacity</code>	500 000	2.31	2.31
<code>NewDirectByteBuffer</code>	300	0.007909092	13.2

`NewDirectByteBuffer`-kutsuprofiili listauksessa 5.9 näyttää lähes yksinomaan roskienkeruuseen liittyviä metodikutsuja alkaen metodista `dvmCollectGarbage`. Tämä

on mittaustekninen ongelma, joka tekee kaikkien muistia varaavien operaatioiden kutsuprofiilit hankaliksi hyödyntää. Siinä missä testin suoritus on saatu mitattua ilman roskienkeruuta *vasteaikamittauksissa*, kutsuprofiilinäytteet kerätään yhtäjaksoisesti useiden sekuntien jaksolta, jonka aikana suhteellisesti suurin osa ajasta vietetään roskienkeruussa. Allokoivia operaatioita ei voi ajaa kuin lyhyen ajan ennen kuin muistia täytyy vapauttaa, jotta se ei loppuisi.

Kenttien käsittely

Kuvaan 5.22 on koottu Javan olioiden ja luokkien kenttien käsittelemiseen käytettävien JNI-metodien vasteajat. Vertailukohtina kuvassa ovat vastaavat operaatiot puhtaasti Java-komponentissa suoritettuina.

Kuvasta nähdään, että staattisten kenttien luku ja kirjoittaminen kestää noin puolet ilmentymäkenttien luvusta ja kirjoittamisesta. Viitetyyppisen ilmentymäkentän lukeminen on vielä hieman hitaampaa kuin muiden ilmentymäkenttien.

Profileista tunnistetaan sama kaava kuin aiemmin: siinä missä staattisten kenttien käsittelyn ainoa merkittävä JNI-lisärasite johtuu virtuaalikoneen tilamuutoksesta, ilmentymäkenttää käsitellessä joudutaan lisäksi purkamaan globaali viite ilmentymään, jonka kenttää käsitellään. `GetObjectField`-operaatiossa joudutaan myös lisäämään kentästä luettu olioviite viitetaulukkoon `addLocalReference`-kutsulla, koska kyseinen viitearvo välitetään natiivikomponentille, ja sen on siksi löydettävä virtuaalikoneen viitetaulukosta.

Sivuutetut operaatiot

Tutkielmassa jätettiin mittaamatta osa JNI-rajapinnan lukuisista metodeista, ja osa mittaustuloksistakin jätettiin lopulta tarkemmin analysoimatta. Keskeinen tutkielman tavoite oli löytää rajapinnan käytön keskeisimmät pullonkaulat, eli sellaiset operaatiot joita saattaa rajapinnan realistisessa käytössä joutua käyttämään usein. Siksi sovelluksen elinkaaren aikana harvoin tai vain erikoistilanteissa kutsuttavat rajapinnan palvelut rajattiin tutkielman ulkopuolelle.

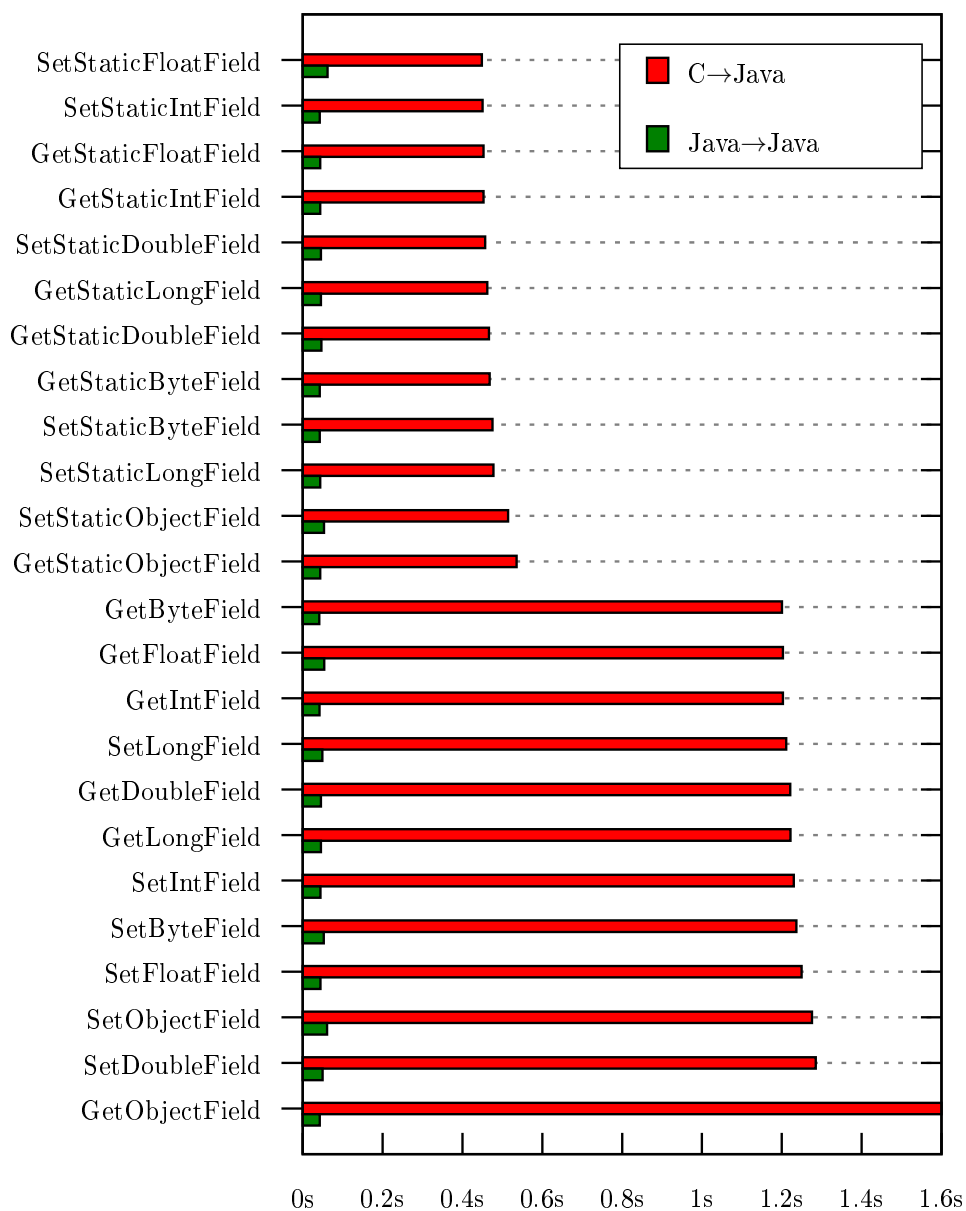
Nämä harvemmin tarvittavat palvelut liittyvät muun muassa eksplisiittiseen Java-viitteiden käsittelyyn, Java-olioiden ja -luokkien erityisominaisuuksien käsittelyyn, reflektioon sekä natiivimetodien rekisteröintiin. Esimerkiksi metodien rekisteröinti tapahtuu tyypillisesti vain kerran sovelluksen suoritusajana. Sen sijaan mahdollisesti oleellisemmat, mutta mittausten yksinkertaistamisen vuoksi pois rajatut palvelut

Listaus 5.9: NewDirectByteBuffer

74.31%	dvmCallMethodV(Thread*, Method const*, Object*, bool, JValue*, std::__va_list)
└─ 73.83%	dvmInterpret(Thread*, Method const*, JValue*)
└─ 72.21%	dalvik_mterp
└─ Dalvik_java_lang_Runtime_gc(unsignedintconst*,JValue*)	
└─ dvmCollectGarbage()	
└─ 72.19%	dvmCollectGarbageInternal(GcSpec const*)
└─ 43.71%	dvmHeapScanMarkedObjects()
└─ 19.92%	dvmHeapSweepUnmarkedObjects(bool, bool, unsigned int*, unsigned int*)
└─ 3.74%	dvmHeapReScanMarkedObjects()
└─ 2.67%	dvmHeapMarkRootSet()
└─ dvmVisitRoots(void(*) (void*, unsignedint, Root Type, void*), void*)	
└─ 1.73%	dvmHeapReMarkRootSet()
└─ dvmVisitRoots(void(*) (void*, unsignedint, Root Type, void*), void*)	
└─ 0.09%	dvmHeapFinishMarkStep()
└─ 0.02%	dvmGetRelativeTimeMsec()
└─ dvmGetRelativeTimeUsec()	
└─ 0.02%	dvmHeapSweepSystemWeaks()
└─ dvmGcDetachDeadInternedStrings(int(*) (void*))	
└─ 0.12%	dalvik_inst
└─ 0.10%	dvmAllocObject
└─ dvmMalloc(unsignedint,int)	
└─ 0.04%	dvmUnlockHeap()
└─ dvmUnlockMutex(pthread_mutex_t*)	
└─ pthread_mutex_unlock	
└─ 0.02%	pthread_mutex_unlock_impl
└─ 0.04%	tryMalloc(unsigned int)
└─ 0.02%	dvmHeapSourceAlloc(unsigned int)
└─ countAllocation(Heap*,voidconst*)	
└─ 0.02%	dvmLockHeap()
└─ dvmTryLockMutex(pthread_mutex_t*)	
└─ 0.04%	common_updateProfile
└─ 0.02%	dvmJitGetTraceAddrThread
└─ getCodeAddrCommon(unsignedshortconst*,bool)	
└─ 0.03%	common_selectTrace
└─ 0.20%	callPrep(Thread*, Method const*, Object*, bool)

liittyvät rinnakkaisohjelmointiin (`MonitorEnter` ja `MonitorExit`) ja poikkeustenkäsitteelyyn. Ne olisivat mielenkiintoinen jatkotutkimuksen aihe.

Kuva 5.22: Kenttien käsittely eri kutsusuunnissa



vasteaika 500000 toistolla

Mittauksia tehtiin myös *allokoiville* rajapinnan palveluille, joilla luodaan uusia Java-olioita. Näitä operaatioita ei kuitenkaan käsitelty systemaattisesti, koska muiden tulosten valossa niidenkään laajamittainen käyttäminen natiivikomponentista tuskin on tehokasta jos se on vältettävissä – viitetyyppisten olioiden käytöstä kun syntyvät keskeisimmät JNI-rajapinnan kustannukset.

5.3 Dalvikin JNI-rajapinnan suorituskykymalli

Seuraavaksi mittaustuloksista rakennetaan karkea malli, joka kuvaa, mistä osatekijöistä JNI-rajapinnan käytön rasitteet syntyvät. Mallista saatavat lukuarvot ovat suhdelukuja, jotka ilmaisevat eri rasitteiden suuruusluokan suhteessa toisiinsa. Mallissa on vain lineaarisia riippuvuuksia, ja *muuttuja*-sarakkeessa ilmaistaan, mikä on riippuvuudessa esiintyvän muuttujan tulkinta.

Kun mallin muuttujiin sijoittaa jonkin konkreettisen JNI-rajapinnan käyttötapauksen arvot, nähdään erottelu kyseisen käyttötapauksen *JNI:n käytöstä* muodostuvan rasitteen jakautumisesta osatekijöihinsä. Kun osatekijöiden rasitteet laskee yhteen, saa tulokseksi JNI:stä johtuva kokonaisrasite. Eri käyttötapauksille saatuja kokonaisrasitteiden vertailulukuja voi myös verrata keskenään.

	rasite	muuttuja	C→Java	Java→C
1	JNI-perusrasite	JNI-kutsujen määrä c	$0.28c$	$0.39c$
2	aliohjelmakutsun perusrasite	aliohjelmakutsujen määrä f	$0.97f$	-
3	virtuaalimetodikutsu	virtuaalikutsujen määrä v	$0.64v$	$0.05v$
4	parametrien välitys	parametrien määrä p	$0.05p$	$0.0085p$
5	viiteparametrien välitys	globaalien viitteiden määrä g	$0.59g$	-
6		lokaalien viitteiden määrä l	$0.27l$	$0.32l$
8	muistialueen kiinnitys ja irroitus	operaatioiden määrä n	$0.6n$	-
9	datan kopiointi	tavujen määrä b	$0.00053b$	-

Malli *ei* kerro käyttötapauksen kokonaisrasitetta *suhteessa JNI:n käytöstä saatuun hyötyyn*. Tämä on hyvin tapauskohtaista, ja tiedon saamiseksi tarvittaisiin tarkat mittaukset siitä, miten paljon tehokkaampaa natiivikomponentin käyttö on verrattuna saman laskennan suorittamiseen Java-koodissa.

Mallista saatuihin tuloksiin tulee suhtautua summittaisena arviona, sillä suoritettavan ohjelman käyttäytymistä kokonaisuutena aidossa suoritusympäristössä ei voi täysin palauttaa ohjelmien alkeisosien itsenäiseen käyttäytymiseen erillään kokonaisuudesta. Mallia voi käyttää osana ohjelmistojen suorituskykytekniikan (SPE)

menetelmiä sovelluksen varhaisen kehitysvaiheen apuna, sillä sen avulla voi saada arvion vasta suunnitteluvaiheessa olevan ohjelmiston suorituskykykäyttäytymisestä.

Malliin ei ole otettu mukaan kaikkia JNI-rajapinnan tarjoamia erikoistoimintoja kuten merkkijonojen merkistömuunnoksia, vaan se keskittyy perusoperaatioihin: aliohjelmauksiin ja taulukkomuotoisen datan välitykseen.

Taulukon kohta *JNI-perusrasite* (1) pitää sisällään JNI-funktion kutsumisesta aiheutuvan kustannuksen, virtuaalikoneen tilanmuutoksen, ja kaikki muut mahdolliset Dalvikin JNI-toteutuksen kustannukset, jotka ovat *yhteisiä kaikille saman kutsusuunnan JNI-operaatioille*. Kutsusuunnassa Java→C ei ole muita operaatioita kuin natiivimetodikutsu, joten aliohjelmauksen perusrasite on mukana JNI-perusrasitteessa.

Sovelletaan mallia esimerkkitapaukseen, jossa suunnassa C→Java kutsutaan void-arvon palauttavaa instanssimetodia virtuaalikutsuna kymmenellä long[]-tyyppisellä parametrilla. C-ohjelmassa kutsu näyttää tältä:

```
(*env)->CallVoidMethod(env, java_object, mid, 11, [...] 110);
```

JNI-kutsuja on tässä yksi, joka on aliohjelmaus ja virtuaalikutsu. Parametreja kutsussa on 12 (JNI-kutsun *env*-parametria ei lasketa parametrimäärään), joista 11 on tässä tapauksessa globaaleja viitteitä. Malliin sijoittamalla saadaan seuraavat arvot.

1	perusrasite	1 · 0.28	0.28
2	aliohjelmaus	1 · 0.97	0.97
3	virtuaalimetodikutsu	1 · 0.65	0.65
4	parametrit	12 · 0.05	0.6
5	globaalit viitteet	11 · 0.59	6.49
yhteensä			8.99

Mallista huomataan, että ehdottomasti suurin osa JNI-operaatioiden kustannuksista syntyy muistinhallintaan liittyvistä operaatioista: viitteiden tallentamisesta viitetaulukkorakenteeseen, niiden noutamisesta sieltä sekä muistialueiden kiinnikytkestä ja kytkemisen poistamisesta. Esimerkiksi melko pienillä syötteillä nämä kustannukset selvästi ylittävät koko syötteen kopioimisen kustannukset. Muistinhallintaan liittyy välillisesti myös kaikkien JNI-operaatioiden suorittama virtuaalikoneen tilamuutos.

Verrattuna normaaliin aliohjelmaukseen yksikielisessä ohjelmassa JNI:n välityksellä tehtäviin aliohjelmauksiin tulee aina myös merkittävä lisärasite, joka johtuu

kutsun epäsuoruudesta, kutsukonvention muuntamisesta kielten välillä, mahdollisten välttämättömien viitteiden käsittelystä sekä edellä mainitusta tilamuutoksesta.

Tulokset voidaan tiivistää näin: tehokas Java Native Interface-rajapinnan käyttö on sellaista, joka saa hyödynnettyä natiivikomponentista mahdollisesti saatavan suorituskykyedun mahdollisimman pienellä määrällä JNI-kutsuja, tallentaen mahdollisimman vähän Java-oliovitteitä pysyvästi natiivikomponenttiin globaalien viitteiden muodossa sekä välittäen ylipäättään lukumääräisesti mahdollisimman vähän Javan muistinhallinnan piirissä olevaan kekkoon allokoituja olioita natiivikomponentille.

Lopputuloksena voidaan todeta, että Androidin Dalvik-suoritusympäristön käyttäytymistä saatiin mitattua kvantitatiivisesti niin, että tuloksista saatiin hyödyllistä ennakkotietoa monikielisten Android-sovellusten käyttäytymisestä. Laaditun mallin avulla sovelluksen suunnittelijan on mahdollista esimerkiksi vertailla kahden vaihtoehdoisen arkkitehtuurimallin suorituskykyäsitteitä ennen kuin sitoutuu jompaankumpaan malliin. Lisäksi tulosten avulla saatiin selville joitakin natiivirajapinnan käyttötapoja, joita tulisi välttää kaikissa rajapintaa hyödyntävissä sovelluksissa.

6 Yhteenveto

Android-sovelluksia on mahdollista rakentaa yhdistelemällä Androidin virtuaalikoneen suorittamia Java-komponentteja natiivikirjastojen komponentteihin, jotka on käännetty suoritussympäristön konekielelle ja joita suoritetaan virtuaalikoneen tavallisen tulkinta- tai käännösprosessin ulkopuolella.

Jotta Java-komponentit ja natiivikomponentit voisivat kutsua toisiaan ja käsitellä toistensa muuttujia ja tietorakenteita, Java-virtuaalikoneet tarjoavat Java Native Interface (JNI) -standardin, joka on vieraskutsurajapinta natiivi- ja Java-komponenttien välillä. Myös Androidin Dalvik- ja ART-virtuaalikoneet toteuttavat JNI-rajapinnan.

Rajapinnan käytöstä aiheutuu kuitenkin erilaisia ylimääräisiä suoritussykyrasitteita verrattuna tavallisiin, samankielisen komponentin sisällä tehtäviin aliohjelmakutsuihin ja tietorakenteita käsitteleviin operaatioihin.

Tutkielmassa tarkasteltiin vasteaika- ja kutsuprofilimittausten avulla Androidin Dalvik-virtuaalikoneen JNI-toteutuksen ominaispiirteitä suoritussyvyn näkökulmasta. Vasteikamittauksissa mitattavia JNI-operaatiota toistettiin silmukassa ja mitattiin suorituksen kokonaisvasteaika. Lisäksi samoja operaatioita tutkittiin näytteenottomenetelmällä, jossa prosessorin tilasta ja erityisesti kutsupinosta otetaan tasan väliajoin näytteitä, joista muodostetaan koko mittausjakson ajalta *kutsuprofiili*. Menetelmiä yhdistelemällä saatiin selville paitsi riippuvuuksia eri parametrien ja kokonaisvasteajan välillä, myös ne Dalvik-virtuaalikoneen sisäisen toteutuksen osat, joiden kesken suoritusaika jakautui.

Virtuaalikoneen käyttäytymisestä saatiin erotettua lineaarisia riippuvuuksia, jotka paljastivat, että virtuaalikoneen muistinhallinta aiheuttaa välillisesti Dalvikin JNI-rajapinnan käytön merkittävimmät rasitteet. Jokaisessa rajapinnan operaatiossa ja jokaisessa sen kautta tehdyssä aliohjelmakutsussa on myös merkittävä yleisrasite, joten kutsujen määrän kasvaessa vasteaika kasvaa nopeasti.

Muistinhallinnan suurimmat rasitteet johtuvat siitä, että Dalvik pitää kirjaa natiivikomponenteille välitetyistä epäsuorista viitteistä tarkoitusta varten rakennetun taulutietorakenteen avulla. Tämä on välttämätöntä, jotta virtuaalikoneen muistinhallinnalla säilyisi tieto Java-suoritussympäristön ulkopuolelle välitetyistä olioviitteistä tilanteessa, jossa roskienkeruu ja muistinhallinnan operaatiot voivat siirrellä olioita muistiosoitteesta toiseen. Jos taas natiivikomponentti käsittelee Java-aulukon tai

merkkijonon muistialuetta suoraan, täytyy kyseiset oliot kiinnikytkeä, mikä estää niiden siirtämisen muistissa. Tästäkin operaatiosta syntyy kustannuksia.

Tutkielman mittaukset tehtiin edellisen sukupolven Android-virtuaalikoneessa Dalvikissa. Käytetty metodologia ja suurin osa mittausohjelmistosta on kuitenkin sellaisenaan siirrettävissä Androidin uudemmalle ART-virtuaalikoneelle. Mittausten toistaminen eri ympäristöversioissa ja laitteissa lisäisi niiden yleistettävyyttä. Vaikka ART-virtuaalikoneen sisäinen toteutus eroaa merkittävästi Dalvikin toteutuksesta, samat yleiset periaatteet pätevät: suuri osa JNI-rajapinnan rasitteista johtuu kaikissa virtuaalikoneissa hyvin todennäköisesti automaattisen muistinhallinnan piirissä olevien komponenttien yhdistämisestä hallitusti natiivikomponenttiin, jossa muistinhallinta on manuaalista. Itse mittaustulokset ja suorituskymalli eivät päde ART-virtuaalikoneeseen, sillä esimerkiksi erilainen roskienkeruumenetelmä sekä JNI-rajapinnan sisäinen toteutus vaikuttavat tietenkin suoraan mitattuihin arvoihin – esimerkiksi tiivistävä roskienkeruu johtaa siihen, että kiinnikytkemisen sijaan merkkijono- ja taulukko-operaatiot ovat aina kopioivia.

Muita mielenkiintoisia kehityskohteita olisi tutkia kattavammin erilaisia JNI-rajapinnan parametrivaihtoehtoja: erityisesti lokaalien ja globaalien viitteiden erottelu osoittautui hyvin oleelliseksi tekijäksi, jota tutkielman mittauksissa ei riittävästi tutkittu. Toinen selkeä puute mittausmenetelmissä oli keskittyminen alkeisoperaatioiden mittaamiseen. Realistisemman ja kokonaisvaltaisemman kuvan rajapinnan käyttäytymisestä saisi, jos mittaushetkeisiin lisättäisiin korkeamman tason käyttötapauksia, jotka edustaisivat paremmin todellisten sovellusten tarpeita, jolloin samassa testissä käytettäisiin useita rajapinnan alkeisoperaatioita. Esimerkiksi suurten datamäärien välittäminen rajapinnan välityksellä komponentista toiseen taulukko- tai tavupuskuritietorakenteiden avulla olisi mielekäs tutkimuskohde. Alkeisoperaatioiden tapauksessa ongelmaksi jää, että kokonaisen ohjelman käyttäytyminen ei palaudu tyhjentävästi siinä käytettyjen osien käyttäytymiseen, mikä johtuu esimerkiksi prosessorin välimuistin ja ennustavien heuristiikkojen kaltaisista vaikeasti ennakoitavista tekijöistä.

Tutkielmasta saatiin kuitenkin oleellista tietoa JNI-rajapinnan käyttäytymisestä, mikä tiivistettiin yksinkertaiseen lineaariseen malliin. Mallia voisi hyödyntää esimerkiksi ohjelmistojen suorituskäytännöksi SPE:tä hyödyntävässä ohjelmistoprojektissa, jossa rajapinnan käyttöä harkitaan suorituskäytännön tavoittelemiseksi. Vaikka konkreettisen Android-sovelluksen lopullinen suorituskäytännön käyttäytyminen voidaan havaita vasta sovellusta itseään tutkimalla, voidaan tutkielman mallin avulla saada

jonkinlainen arvio rajapinnan pullonkauloista eri arkkitehtuurivaihtoehtoisissa. Eri-tyisesti JNI-rajapinnan patologisten käyttötapauksen välttämiseen saatiin lisävalaistusta. Sovelluksen hyvää suorituskykyä edistää rajapintakutsujen määrän ja rajapinnan kautta välitettyjen viiteparametrien määrän rajoittaminen sekä jonkin ver-
ran myös natiivikomponentista Java-komponenttiin suuntautuvien metodikutsujen välttäminen.

Lähteet

- ABL97 Ammons, G., Ball, T. ja Larus, J. R., Exploiting hardware performance counters with flow and context sensitive profiling. *Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation*, PLDI '97, New York, NY, USA, 1997, ACM, sivut 85–96, URL <http://doi.acm.org/10.1145/258915.258924>.
- ADMT88 Appel, A., Duba, B. F., Macqueen, D. B. ja Tolmach, A. P., Profiling in the presence of optimization and garbage collection. Tekninen raportti CS-TR-197-88, Princeton University, Department of Computer Science, Princeton, NJ, USA, Nov. 1988.
- aos16 Android open source project, *Android open source project—Downloading and building—Preparing to build*, 2016. Tekninen dokumentaatio. <https://source.android.com/source/building.html>. [20.8.2016]
- App16 Apple Inc., *Using Swift with Cocoa and Objective-C*, 2016. <https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/index.html>. [20.8.2016]
- ARM10 ARM Ltd., *Cortex-A8 Technical Reference Manual, Revision r3p2*, 2010.
- ARM13 ARM Ltd., *Cortex A Series Programmers Guide, Version 4.0*, 2011–2013.
- AS11 Absar, J. ja Shekhar, D., Eliminating partially-redundant array-bounds check in the Android Dalvik JIT compiler. *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java*, PPPJ '11, New York, NY, USA, 2011, ACM, sivut 121–128, URL <http://doi.acm.org/10.1145/2093157.2093175>.
- BMG⁺08 Blackburn, S. M., McKinley, K. S., Garner, R., Hoffmann, C., Khan, A. M., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S. Z., Hirzel, M., Hosking, A., Jump, M., Lee, H., Eliot, J., Moss, B., Phansalkar, A., Stefanovic, D., VanDrunen, T., von Dincklage, D. ja

- Wiedermann, B., Wake Up and Smell the Coffee: Evaluation Methodology for the 21st Century. *Communications of the ACM*. Invited paper. CACM Research Highlights.
- BSS⁺09 Batyuk, L., Schmidt, A.-D., Schmidt, H.-G., Camtepe, A. ja Albayrak, S., Developing and benchmarking native Linux applications on Android. Teoksessa *MobileWireless Middleware, Operating Systems, and Applications*, Bonnin, J.-M., Giannelli, C., Magedanz, T., Akan, O., Bellavista, P., Cao, J., Dressler, F., Ferrari, D., Gerla, M., Kobayashi, H., Palazzo, S., Sahni, S., Shen, X. S., Stan, M., Xiaohua, J., Zomaya, A. ja Coulson, G., toimittajat, osa 7 sarjasta *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Berlin Heidelberg, 2009, sivut 381–392.
- CHCC13 Chang, Y.-M., Hsiu, P.-C., Chang, Y.-H. ja Chang, C.-W., A resource-driven DVFS scheme for smart handheld devices. *ACM Transactions on Embedded Computing Systems*, 13,3(2013), sivut 53:1–53:22. URL <http://doi.acm.org.libproxy.helsinki.fi/10.1145/2539036.2539049>.
- CHL⁺12 Cho, M., Hwang, S. J., Lee, H. J., Kim, M. ja Kim, S. W., Androscope for detailed performance study of the Android platform and its applications. *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, jan. 2012, sivut 408 –409.
- Coc63 Cochran, W. G., *Sampling techniques*. Wiley, New York (NY), 1963.
- CW76 Curnow, H. J. ja Wichmann, B. A., A synthetic benchmark. *The Computer Journal*, 19,1(1976), sivut 43–49. URL <http://comjnl.oxfordjournals.org/content/19/1/43.abstract>.
- DSZ11 Du, J., Sehrawat, N. ja Zwaenepoel, W., Performance profiling of virtual machines. *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, New York, NY, USA, 2011, ACM, sivut 3–14, URL <http://doi.acm.org/10.1145/1952682.1952686>.
- Dun07 Dunlavey, M., Performance tuning with instruction-level cost derived from call-stack sampling. *ACM SIGPLAN Notices*, 42,8(2007), sivut 4–8. URL <http://doi.acm.org/10.1145/1294297.1294298>.

- Fer78 Ferrari, D., *Computer systems performance evaluation*. Prentice-Hall, Englewood Cliffs (NJ), 1978.
- GKM82 Graham, S. L., Kessler, P. B. ja Mckusick, M. K., Gprof: A call graph execution profiler. *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*, SIGPLAN '82, New York, NY, USA, 1982, ACM, sivut 120–126, URL <http://doi.acm.org.libproxy.helsinki.fi/10.1145/800230.806987>.
- Gla97 Glass, R. L., Software runaways—some surprising findings. *ACM SIGMIS Database*, 28,3(1997), sivut 16–19. URL <http://doi.acm.org/10.1145/272657.272687>.
- Goe05 Goetz, B., Java theory and practice: Anatomy of a flawed microbenchmark. *IBM developerWorks*. <http://www.ibm.com/developerworks/library/j-jtp02225/>. [20.8.2016]
- Goo16a Google, Android dashboards, 2016. <https://developer.android.com/about/dashboards/index.html#Platform>. [20.8.2016]
- Goo16b Google, *Android Native Development Kit*, 2016. <http://developer.android.com/tools/sdk/ndk/index.html>. [20.8.2016]
- Goo16c Google, *Android SDK API Reference*, 2016. <http://developer.android.com/reference/packages.html>. [20.8.2016]
- Goo16d Google, Art and Dalvik. <https://source.android.com/devices/tech/dalvik/>. [20.8.2016]
- Goo16e Google, *JNI Tips*, 2016. <http://developer.android.com/training/articles/perf-jni.html>. [20.8.2016]
- Goo16f Google, *Profiling with Traceview and dmtracedump*, 2016. <http://developer.android.com/tools/debugging/debugging-tracing.html>. [20.8.2016]
- Goo16g Google, *Verifying App Behavior on the Android Runtime (ART)*, 2016. <https://developer.android.com/guide/practices/verifying-apps-art.html>. [20.8.2016]

- HAKB07 Harrison, C., Amento, B., Kuznetsov, S. ja Bell, R., Rethinking the progress bar. *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, New York, NY, USA, 2007, ACM, sivut 115–118, URL <http://doi.acm.org/10.1145/1294211.1294231>.
- HG07 Hirzel, M. ja Grimm, R., Jeannie: granting Java native interface developers their wishes. *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, OOPSLA '07, New York, NY, USA, 2007, ACM, sivut 19–38, URL <http://doi.acm.org/10.1145/1297027.1297030>.
- HLTW03 Heckmann, R., Langenbach, M., Thesing, S. ja Wilhelm, R., The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 91,7(2003), sivut 1038–1054.
- HSDH04 Hauswirth, M., Sweeney, P. F., Diwan, A. ja Hind, M., Vertical profiling: understanding the behavior of object-oriented applications. *ACM SIGPLAN Notices*, 39,10(2004), sivut 251–269. URL <http://doi.acm.org/10.1145/1035292.1028998>.
- Knu71 Knuth, D. E., An empirical study of FORTRAN programs. *Software: Practice and Experience*, 1,2(1971), sivut 105–133. URL <http://dx.doi.org/10.1002/spe.4380010203>.
- Knu74 Knuth, D. E., Structured programming with go to statements. *ACM Comput. Surv.*, 6,4(1974), sivut 261–301. URL <http://doi.acm.org/10.1145/356635.356640>.
- Lia99 Liang, S., *Java Native Interface: Programmer's Guide and Reference*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, ensimmäinen painos, 1999.
- LJ10 Lee, S. ja Jeon, J. W., Evaluating performance of Android platform using native C for embedded systems. *Control Automation and Systems (ICCAS), 2010 International Conference on*. IEEE, Oct. 2010, sivut 1160 –1163.
- LLDW11 Lin, C.-M., Lin, J.-H., Dow, C.-R. ja Wen, C.-M., Benchmark Dalvik and native code for Android system. *Innovations in Bio-inspired*

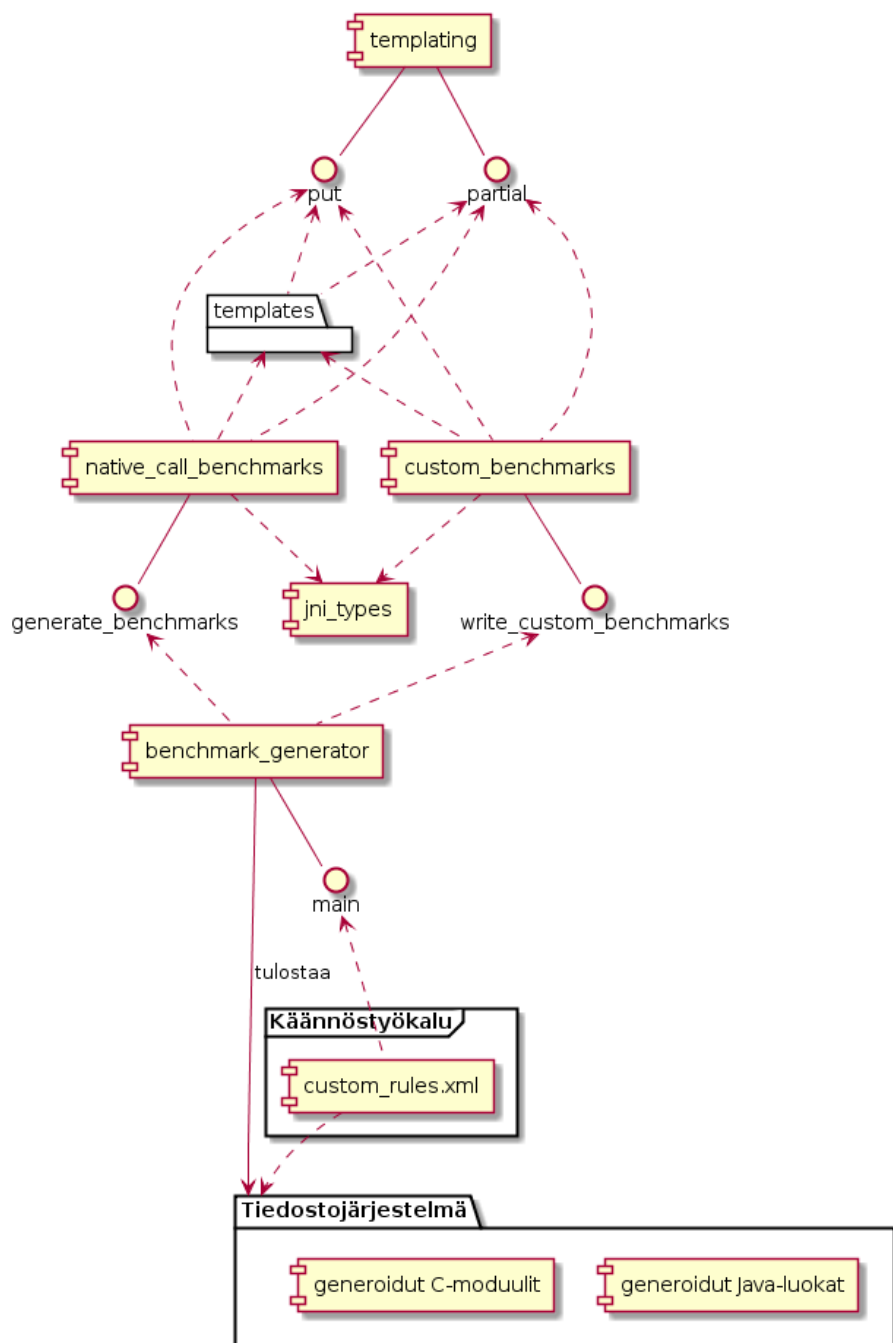
Computing and Applications (IBICA), 2011 Second International Conference on. IEEE, Dec. 2011, sivut 320–323.

- MDHS10 Mytkowicz, T., Diwan, A., Hauswirth, M. ja Sweeney, P. F., Evaluating the accuracy of Java profilers. *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation, PLDI '10*, New York, NY, USA, 2010, ACM, sivut 187–197, URL <http://doi.acm.org/10.1145/1806596.1806618>.
- Mis16 Misc., *Linux kernel profiling with perf*, elokuu 2016. <https://perf.wiki.kernel.org/index.php/Tutorial>. [20.8.2016]
- Ope12 Open Handset Alliance, Android open source project, marraskuu 2012.
- RS06 Reppy, J. ja Song, C., Application-specific foreign-interface generation. *Proceedings of the 5th international conference on Generative programming and component engineering, GPCE '06*, New York, NY, USA, 2006, ACM, sivut 49–58, URL <http://doi.acm.org/10.1145/1173706.1173714>.
- SCEG08 Shi, Y., Casey, K., Ertl, M. A. ja Gregg, D., Virtual machine showdown: Stack versus registers. *ACM Transactions on Architecture and Code Optimization*, 4,4(2008), sivut 2:1–2:36. URL <http://doi.acm.org/10.1145/1328195.1328197>.
- Smi90 Smith, C. U., *Performance engineering of software systems*. Addison-Wesley, Reading (MA), 1990.
- SR08 Stevens, W. R. ja Rago, S. A., *Advanced Programming in the UNIX Environment, Second Edition*. Addison-Wesley Professional, toinen painos, 2008.
- SS96 Saavedra, R. H. ja Smith, A. J., Analysis of benchmark characteristics and benchmark performance prediction. *ACM Transactions on Computer Systems*, 14,4(1996), sivut 344–384. URL <http://doi.acm.org/10.1145/235543.235545>.
- Tri82 Trivedi, K. S., *Probability and statistics with reliability, queuing, and computer science applications*. Prentice-Hall, Englewood Cliffs (NJ), 1982.

- VM09 Vieira, M. ja Madeira, H., From performance to dependability benchmarking: A mandatory path. Teoksessa *Performance Evaluation and Benchmarking*, Nambiar, R. ja Poess, M., toimittajat, osa 5895 sarjasta *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2009, sivut 67–83, URL http://dx.doi.org/10.1007/978-3-642-10424-4_6.
- WEE⁺08 Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J. ja Stenström, P., The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7,3(2008), sivut 36:1–36:53. URL <http://doi.acm.org/10.1145/1347375.1347389>.
- WFP07 Woodside, M., Franks, G. ja Petriu, D. C., The future of software performance engineering. *Future of Software Engineering*, FOSE '07, Washington, DC, USA, 2007, IEEE Computer Society, sivut 171–187, URL <http://dx.doi.org/10.1109/FOSE.2007.32>. ICSE 2007-konferenssin yhteydessä järjestetty konferenssi, Minneapolis, MN, USA, 23. — 25.5.2007.
- WPC⁺11 Wang, C.-S., Perez, G., Chung, Y.-C., Hsu, W.-C., Shih, W.-K. ja Hsu, H.-R., A method-based ahead-of-time compiler for Android applications. *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*, CASES '11, New York, NY, USA, 2011, ACM, sivut 15–24, URL <http://doi.acm.org/10.1145/2038698.2038704>.
- WS98 Williams, L. G. ja Smith, C. U., Performance evaluation of software architectures. *Proceedings of the 1st international workshop on Software and performance*, WOSP '98, New York, NY, USA, 1998, ACM, sivut 164–177, URL <http://doi.acm.org/10.1145/287318.287353>.
- Yag13 Yaghmour, K., *Embedded Android*. O'Reilly Media, early release- painos, tammikuu 2013.

Liite 1. Mittausten generointi ja käsittely

Tässä liitteessä kuvaillaan suorituskykytestien generoimiseen ja testitulosten analysoimiseen laadittuja Python-ohjelmia.



Kuva 1: Benchmark Generator: komponentit

Mittauksetien koodingeneroija Benchmark Generator

Kuvan 1 komponenttikaaviossa näkyy suorituskykytestit generoivan **Benchmark Generator**-ohjelman perusarkkitehtuuri. Kuvan alaosasta nähdään, että Ant-käännöstyökalu on konfiguroitu kutsumaan automaattisesti **benchmark_generator**-päämoduulia, ja generoidut lähdekooditiedostot tulevat näin osaksi normaaleja Androidin SDK- ja NDK- käännösprosesseja.

Lähdekoodien generointi perustuu seuraavaan kahteen suunnitteluratkaisuun. Ensinnäkin generointikoodin syötteenä ovat **jni_types**-moduulissa määritellyt tietorakenteet, jotka sisältävät tarvittavat tiedot kaikista Java-ohjelmoinnin tyypeistä ja niiden vastinpareista C-kielessä.

Toinen ratkaisu on käsitellä Java- ja C-koodia merkkijonoina eikä esimerkiksi rakenteellisina syntaksipuina. Monimutkaisia lähdekoodimerkkijonoja on vältetty upottamasta suoraan pythonkielisen logiikan keskelle. Lähdekoodin generoinnin pohjina toimivat **templates**-pakkauksesta löytyvät aihiot. Ne ovat kohdekielistä koodia sisältäviä merkkijonoja, jonka sisällä olevat nimetyt paikkamerkit korvataan vaihtuvilla arvoilla.

Seuraavassa esimerkissä näkyy **jni_types**-moduulin pythonkielinen **dict**-tyyppimääritelmä Javan **int**-primitiivityypille.

```
{
    'symbol'      : 'i',
    'java'        : 'int',
    'c'           : 'jint',
    'c-literal'   : '102',
    'java-literal': '102',
    'jvm-desc'    : 'I',
    'representative' : True,
    'byte_count'  : '4'
},
```

Tietueen kentistä **symbol** on tyyppin sisäinen tunniste generointikoodissa. Kentät **java** sekä **c** ovat tyyppin nimet molemmissa kielissä, ja **java-literal** sekä **c-literal** sisältävät tyyppiä vastaavan literaalin, jonka voi aina lisätä lähdekoodiin, kun tarvitaan kyseisen tyyppistä arvoa. Kenttä **jvm-desc** on Java-tavukoodissa ja joissain JNI-kutsuissa käytetty tyyppimääritys. Kentän **representative** arvon perusteella voidaan alustavien mittauksen jälkeen välttää turhien testien aikaa vievä suoritus – esimerkiksi **int**- ja **char**-tyyppien välillä ei huomattu mitään eroa, joten mittauksissa voidaan käyttää vain **int**-tyyppisiä testejä edustamaan molempia.

Generointiin käytettävät aihiot näyttävät seuraavalta. Esimerkki on tiedostosta `templates/c_nativemethod.py`.

```
t_run_method = """
JNIEXPORT <% return_type %> JNICALL
Java_<% packagename %>_<% classname %>_<% function %>
(JNIEnv *env, <% parameters %>) {
    <% parameter_declarations %>;
    <% parameter_initialisations %>;
    <% prebody %>
    <% body %>
}

"""
```

Template-kielen paikkamerkkeinä tulkittavat osat erotetaan kirjaimellisesti tulkittavasta osasta erottimilla `<%` ja `%>`, joille ei ole varattua merkitystä kohdekielissä. Kyseisten merkintöjen käyttöä aihioissa esimerkiksi kohdekielen merkkijonoliteraalien sisällä ei tueta.

Aihion voi täydentää valittujen parametrien arvoilla `templating`-moduulin funktioilla `put` ja `partial`. Funktio `put` korvaa aihion kaikki paikkamerkit samannimisten nimettyjen parametrien arvoilla poistaen puuttuvia argumentteja vastaavat paikkamerkit. Funktio `partial` jättää puuttuvien argumenttien paikkamerkinnyt paikoilleen, mikä helpottaa erikoistuneiden aihiodien johtamista samasta perusaihiosta. Funktiota `put` käytetään kuten seuraavassa esimerkissä.

```
'code' : put(
    arrays.t_read,
    declare_idx = 'int_idx;',
    declare_variables = java_declarations,
    variable_in = '%sIn' % _type['java'],
    array_variable = '%sArr' % _type['java'],
    element_literal = _type['java-literal']
),
'finished' : 'persistentValue=localPersistentValue;'

[...]
```

Molemmat funktiot on helppo toteuttaa, kun tukena on seuraavat Pythonin sanakirjaluokasta periytyvät kaksi aliluokkaa.

```
class PartialDict(dict):
    def __missing__(self, key):
        return "<% " + key + " %>"

class PurgeDict(dict):
    def __missing__(self, key):
        return ""
```

Kun `put`-kutsun nimetyt argumentit tallennetaan `PurgeDict`-olioon, mahdollisesti puuttuvien avainten arvoiksi tulee automaattisesti tyhjä. Vastaavasti `partial`-kutsun argumentit tallennettuna `PartialDict`-olioon toimivat niin, että paikkamerkin puuttuva arvo korvataan samanlaisella paikkamerkillä.

Mittaustulosten käsittelijä `Benchmark Analyzer`

NativeBenchmark-sovelluksen näkökulmasta suoritettavilla testeillä ei ole mitään suhdetta toisiinsa, vaan jokainen suoritetaan toisistaan riippumatta ja jokaisesta mittauksesta tallennetaan mahdollisimman täydelliset parametritiedot tekstitiedostoon. Vasteaikamittauksissa tiedostoon tallennetaan myös mittaustulos eli vasteaika, kutsuprofilin tapauksessa sen sijaan tiedostopolku `perf record`-työkalun tulostamaan näytetiedostoon.

Koska kaikki mittausten parametrit on tallennettu rakenteisesti tiedostoon, *Benchmark Analyzer* -ohjelman tehtäväksi jää yhdistellä jälkikäteen saman parametrin vaihtelevat arvot yhdeksi mittaussarjaksi ja piirtää tulokset kuvaajiksi `gnuplot`-ohjelmalla sekä tulostaa ne taulukoiksi tekstitiedostoihin. Analyysiohjelmalla on myös käytössään `jni_types`-moduulin tyyppimäärytykset.

Liite 2. Muutokset Android-alustaan

Tässä liitteessä luetellaan mittauksia varten tehdyt muutokset Android Open Source Projectin julkaisemaan lähdekoodiversioon Android-järjestelmästä. Muutokset vaaditaan, jotta Linuxin `perf`-työkalu pystyy tulkitsemaan oikein kutsupinonäytteet ja tulostamaan aliohjelmista symboliset lähdekoodissa olevat nimet pelkkien muistiosoitteiden sijaan.

Muutokset on tehty Androidin versioon `android-4.1.2_r1` build-tunnisteella `JZ054K`. Muutokset näytetään `git`-työkalun esitysmuodossa.

build

Symbolisten nimien selvitys vaatii binääreihin `build-id` -tunnisteen.

```
project build/
diff --git a/core/combo/arch/arm/armv7-a-neon.mk \
      b/core/combo/arch/arm/armv7-a-neon.mk
index 32273ff..759e626 100644
--- a/core/combo/arch/arm/armv7-a-neon.mk
+++ b/core/combo/arch/arm/armv7-a-neon.mk
@@ -22,4 +22,4 @@ arch_variant_cflags := \
     -mfpu=neon

     arch_variant_ldflags := \
--Wl,--fix-cortex-a8
+ -Wl,--fix-cortex-a8 -Wl,--build-id
```

dalvik

`Perf`-työkalu osaa tulkita vain `arm`-konekielen kutsupinoja. Oletuskieli on `Thumb`.

```
project dalvik/
diff --git a/Android.mk b/Android.mk
index 73ec342..64bb6cc 100644
--- a/Android.mk
+++ b/Android.mk
@@ -14,6 +14,8 @@

LOCAL_PATH := $(call my-dir)
```

```
+LOCAL_ARM_MODE := arm
+
subdirs := $(addprefix $(LOCAL_PATH)/,$(addsuffix /Android.mk, \
libdex \
vm \
```

device

Otettu käyttöön itse käännetty kernel-versio.

```
project device/samsung/crespo/
diff --git a/kernel b/kernel
deleted file mode 100644
index 825e50d..0000000
Binary files a/kernel and /dev/null differ
diff --git a/kernel b/kernel
new file mode 120000
index 0000000..0347ad0
--- /dev/null
+++ b/kernel
@@ -0,0 +1 @@
+/home/tituomin/droid/linux-kernel/arch/arm/boot/zImage
\ No newline at end of file
```

perf

Otetaan perf-työkalu mukaan käännökseen.

```
diff --git a/Android.mk b/Android.mk
index 9109c96..4e9a795 100644
--- a/Android.mk
+++ b/Android.mk
@@ -119,3 +119,3 @@ LOCAL_C_INCLUDES += $(LOCAL_PATH)/host-$(HOST_OS)-fixup
LOCAL_MODULE := libperf
-LOCAL_MODULE_TAGS := eng
+LOCAL_MODULE_TAGS := optional

@@ -156,3 +156,3 @@ LOCAL_C_INCLUDES := external/elfutils external/elfutils/libelf \
external/elfutils
LOCAL_MODULE := libperf
-LOCAL_MODULE_TAGS := eng
```

```
+LOCAL_MODULE_TAGS := optional

@@ -167,3 +167,3 @@ include $(CLEAR_VARS)
    LOCAL_MODULE := perfhost
-LLOCAL_MODULE_TAGS := eng
+LOCAL_MODULE_TAGS := optional

@@ -234,3 +234,3 @@ include $(CLEAR_VARS)
    LOCAL_MODULE := perf
-LLOCAL_MODULE_TAGS := eng
+LOCAL_MODULE_TAGS := optional
```

su

Muokattu su-ohjelmaa sallimaan mittausohjelmiston su-kutsu ilman käyttöoikeus-tarkistuksia.

```
project system/extras/
diff --git a/su/su.c b/su/su.c
index f87f073..4312c07 100644
--- a/su/su.c
+++ b/su/su.c
@@ -45,11 +45,12 @@ int main(int argc, char **argv)
    int uid, gid, myuid;

    /* Until we have something better, only root and the shell can use su. */
-   myuid = getuid();
-   if (myuid != AID_ROOT && myuid != AID_SHELL) {
-       fprintf(stderr, "su: uid %d not allowed to su\n", myuid);
-       return 1;
-   }
+   /* commented out for thesis purposes */
+   /* myuid = getuid(); */
+   /* if (myuid != AID_ROOT && myuid != AID_SHELL) { */
+   /*     fprintf(stderr, "su: uid %d not allowed to su\n", myuid); */
+   /*     return 1; */
+   /* } */

    if(argc < 2) {
        uid = gid = 0;
```

LIITE 3. Mittausohjelmiston lähdekoodi

Timo Tuominen

Tiedostot

NativeBenchmark	3
Java-komponentit	3
ApplicationState.java	3
ApplicationStateListener.java	4
BenchmarkController.java	4
Benchmark.java	7
BenchmarkParameter.java	8
BenchmarkRegistry.java	13
BenchmarkResult.java	14
BenchmarkRunner.java	15
BenchmarkSelector.java	27
Init.java	34
L.java	35
LogAccess.java	35
measuringtool/AllocatingBenchmarkLongRunningWrapper.java	36
measuringtool/AllocatingBenchmarkShortRunningWrapper.java	37
measuringtool/AllocatingBenchmarkWrapper.java	38
measuringtool/BasicOption.java	39
measuringtool/CommandlineTool.java	39
measuringtool/JavaSystemNanoResponseTimeRecorder.java	42
measuringtool/LinuxPerfRecordTool.java	43
measuringtool/MeasuringOption.java	44
measuringtool/MeasuringTool.java	44
measuringtool/MockCommandlineTool.java	50
measuringtool/OptionSpec.java	50
measuringtool/PlainRunner.java	51
measuringtool/ResponseTimeRecorder.java	52
measuringtool/RunningWrapper.java	53
MockObject.java	54
ShellEnvironment.java	55
SocketCommunicator.java	57
ToolConfig.java	61
Utils.java	64
Python-komponentit (koodingenerointi)	67
benchmark_generator.py	67
jni_types.py	74
make_benchmarks.py	78
make_custom_benchmarks.py	80
templates/arrays.py	91

templates/c_jni_function.py	92
templates/c_module.py	92
templates/c_nativemethod.py	95
templates/__init__.py	96
templates/java_benchmark.py	96
templates/java_counterparts.py	98
templates/java_registry_init.py	99
templates/loop_code.py	99
templating.py	101
Python-komponentit (analyysi)	103
/analysis.py	103
/datafiles.py	104
/gnuplot.py	107
/plot_data.py	113
/textualtable.py	133

NativeBenchmark

Java-komponentit

Hakemistossa `src/fi/helsinki/cs/tituomin/nativebenchmark/`.

ApplicationState.java

```
1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.content.res.Resources;
4
5 public interface ApplicationState {
6     public void updateState(State state);
7
8     public void updateState(State state, String message);
9
10    public boolean userWantsToRetry(Exception exception);
11
12    public DetailedState getState();
13
14    public Resources getResources();
15
16    public static enum State {
17        INITIALISED(R.string.app_name),
18        MEASURING_STARTED(R.string.measuring_started),
19        INTERRUPTING(R.string.interrupting),
20        INTERRUPTED(R.string.interrupted),
21        MILESTONE(R.string.measuring_milestone),
22        ERROR(R.string.error),
23        INIT_FAIL(R.string.error),
24        MEASURING_FINISHED(R.string.measuring_finished);
25
26        public final int stringId;
27
28        State(int stringId) {
29            this.stringId = stringId;
30        }
31    }
32
33    public class DetailedState {
34        public State state;
35        public String message;
36        private ApplicationState parent;
37
38        public DetailedState(ApplicationState parent) {
39            this.parent = parent;
40            this.state = null;
41            this.message = null;
42        }
43
44        public DetailedState(ApplicationState parent, DetailedState d) {
45            this.parent = parent;
46            this.state = d.state;
47            this.message = d.message;
```



```

48     }
49
50     public String toString() {
51         Resources resources = parent.getResources();
52         if (this.state == null) {
53             return "<unknown state>";
54         }
55         String type = resources.getString(this.state.stringId);
56         return String.format(
57             "%s%s", type,
58             (this.message != null) ? " " + this.message : "");
59     }
60 }
61 }

```

ApplicationStateListener.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 public interface ApplicationStateListener {
4
5     public void stateUpdated(ApplicationState.DetailedState state);
6 }

```

BenchmarkController.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.app.ActivityManager;
4 import android.content.Context;
5 import android.content.res.Resources;
6 import android.os.PowerManager;
7 import android.util.Log;
8
9 import java.io.File;
10 import java.io.IOException;
11
12 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
13
14 public class BenchmarkController implements ApplicationState {
15
16     public BenchmarkController(Context aContext, File dataDir) {
17         this.detailedState = new ApplicationState.DetailedState(this);
18         this.dataDir = dataDir;
19         this.listeners = new Listeners();
20
21         PowerManager pm = (PowerManager) aContext.getSystemService(Context
22             .POWER_SERVICE);
23         wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,

```

```

24         "Benchmarking");
25     ActivityManager am = (ActivityManager) aContext.getSystemService
26         (Context.ACTIVITY_SERVICE);
27     this.resources = aContext.getResources();
28     int memoryClass = am.getLargeMemoryClass();
29     Log.v("Selector", "Memory size " + Runtime.getRuntime().maxMemory());
30     Log.v("onCreate", "memoryClass:" + Integer.toString(memoryClass));
31
32 }
33
34 public Resources getResources() {
35     return this.resources;
36 }
37
38 public void updateState(ApplicationState.State state) {
39     updateState(state, null);
40 }
41
42 public void updateState(ApplicationState.State state, String message) {
43     synchronized (this) {
44         this.detailedState.state = state;
45         this.detailedState.message = message;
46
47         switch (state) {
48             case MEASURING_STARTED:
49                 try {
50                     LogAccess.start(dataDir);
51                 } catch (IOException e) {
52                     this.detailedState.state = ApplicationState.State.ERROR;
53                     this.detailedState.message = "Could not initialize " +
54                         "log file.";
55                     Log.e(TAG, this.detailedState.message);
56                 }
57                 wakeLock.acquire();
58                 break;
59             case ERROR:
60             case INTERRUPTED:
61             case MEASURING_FINISHED:
62                 if (wakeLock.isHeld()) {
63                     wakeLock.release();
64                 }
65                 LogAccess.end();
66             case INITIALISED:
67             case INIT_FAIL:
68             case MILESTONE:
69                 if (this.listeners.milestoneListener != null) {
70                     this.listeners.milestoneListener.stateUpdated(this
71                         .detailedState);
72                 }
73         }
74     }
75 }
76
77 public ApplicationState.DetailedState getState() {
78     synchronized (this) {
79         return new ApplicationState.DetailedState(this, this.detailedState);
80     }
81 }
82
83 public boolean userWantsToRetry(Exception e) {

```

```

84     return false;
85 }
86
87 private class BenchRunnable implements Runnable {
88     BenchRunnable(BenchmarkRunner runner, ToolConfig configuration) {
89         this.runner = runner;
90         this.configuration = configuration;
91     }
92
93     public void run() {
94         this.runner.runBenchmarks(
95             BenchmarkController.this,
96             this.configuration,
97             dataDir);
98     }
99
100    private BenchmarkRunner runner;
101    private ToolConfig configuration;
102 }
103
104 public void startMeasuring(BenchmarkRunner runner, ToolConfig
105     configuration) {
106     String message = null;
107     if (this.resources.getString(R.string.app_dirty).equals("1")) {
108         message = this.resources.getString(R.string.warning_changed);
109     }
110
111     BenchmarkRunner.BenchmarkSet set = configuration.getBenchmarkSet();
112     runner.setBenchmarkSet(set);
113     measuringThread = new Thread(new BenchRunnable(runner, configuration));
114     if (message != null) {
115         this.updateState(ApplicationState.State.MEASURING_STARTED, message);
116     } else {
117         this.updateState(ApplicationState.State.MEASURING_STARTED);
118     }
119     measuringThread.start();
120 }
121
122 public void interruptMeasuring() {
123     MeasuringTool.userInterrupt();
124     measuringThread.interrupt();
125 }
126
127 public void addListener(ApplicationStateListener listener,
128     ApplicationState.State state) {
129     if (state == ApplicationState.State.MILESTONE) {
130         this.listeners.milestoneListener = listener;
131     }
132 }
133
134 public void removeListeners() {
135     this.listeners = null;
136 }
137
138 private class Listeners {
139     public ApplicationStateListener milestoneListener;
140
141     public Listeners() {
142         milestoneListener = null;
143     }

```

```

144     }
145
146     private Listeners listeners;
147     private ApplicationState.State state;
148     private String message;
149     private ApplicationState.DetailedState detailedState;
150     private PowerManager.WakeLock wakeLock;
151     private File dataDir;
152     private Thread measuringThread;
153     private static final String TAG = "BenchmarkController";
154     private Resources resources;
155
156 }

```

Benchmark.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.os.Process;
4
5 public abstract class Benchmark implements Runnable {
6     protected abstract void runInternal();
7
8     public abstract String from();
9
10    public abstract String to();
11
12    public abstract String description();
13
14    public abstract String id();
15
16    public abstract int sequenceNo();
17
18    public abstract boolean isAllocating();
19
20    public abstract boolean isNonvirtual();
21
22    public abstract boolean dynamicParameters();
23
24    public abstract boolean representative();
25
26    public long repetitionsLeft;
27
28    public void run() {
29        Process.setThreadPriority(-5);
30        runInternal();
31    }
32
33    protected BenchmarkParameter benchmarkParameter;
34    protected long repetitions;
35
36    public void init(BenchmarkParameter bp) {
37        repetitionsLeft = 0;
38        benchmarkParameter = bp;
39        repetitions = -1;

```

```

40     }
41
42     public void setRepetitions(long reps) {
43         if (reps < 1) {
44             return;
45         }
46         repetitions = reps;
47         BenchmarkRegistry.setRepetitions(reps);
48     }
49 }

```

BenchmarkParameter.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.content.pm.PermissionInfo;
4 import android.util.Log;
5
6 import java.nio.ByteBuffer;
7 import java.util.Arrays;
8 import java.util.Iterator;
9 import java.util.NoSuchElementException;
10
11 import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.JavaCounterparts;
12
13 // Important! Imported as an example class.
14
15 public class BenchmarkParameter implements Iterable<Integer> {
16
17     private native int initReturnvalues(int size, MockObject o);
18
19     private native void freeReturnvalues();
20
21     public void setUp() {
22         initReturnvalues(index * DEFAULTSIZE, mockObjectInstance);
23         JavaCounterparts.initParams(this);
24     }
25
26     public void tearDown() {
27         freeReturnvalues();
28     }
29
30     public static final int DEFAULTSIZE = 64;
31     public static final int RANGE = 8;
32     public static final int MAXSIZE = DEFAULTSIZE * RANGE;
33
34     // The byte buffer has to hold longs and doubles too.
35     public static final int NIO_MAXSIZE = DEFAULTSIZE * RANGE * 8;
36
37     public static MockObject mockObjectInstance = new MockObject();
38
39     public MockObject retrieveMockObject() {
40         return mockObjectInstance;
41     }
42

```

```

43 public BenchmarkParameter() {
44     index = 1;
45     if (!generated) {
46         generateAll();
47         generated = true;
48     }
49
50     for (int i = 0; i < RANGE + 1; i++) {
51         if (STRINGS[i] == null) {
52             Log.v("Parameter", "STRINGS is null at " + i);
53         }
54         if (OBJECTS[i] == null) {
55             Log.v("Parameter", "OBJECTS is null at " + i);
56         }
57         if (THROWABLES[i] == null) {
58             Log.v("Parameter", "THROWABLES is null at " + i);
59         }
60         if (BOOLEAN_ARRAYS[i] == null) {
61             Log.v("Parameter", "BOOLEAN_ARRAYS is null at " + i);
62         }
63         if (BYTE_ARRAYS[i] == null) {
64             Log.v("Parameter", "BYTE_ARRAYS is null at " + i);
65         }
66         if (CHAR_ARRAYS[i] == null) {
67             Log.v("Parameter", "CHAR_ARRAYS is null at " + i);
68         }
69         if (DOUBLE_ARRAYS[i] == null) {
70             Log.v("Parameter", "DOUBLE_ARRAYS is null at " + i);
71         }
72         if (FLOAT_ARRAYS[i] == null) {
73             Log.v("Parameter", "FLOAT_ARRAYS is null at " + i);
74         }
75         if (INT_ARRAYS[i] == null) {
76             Log.v("Parameter", "INT_ARRAYS is null at " + i);
77         }
78         if (LONG_ARRAYS[i] == null) {
79             Log.v("Parameter", "LONG_ARRAYS is null at " + i);
80         }
81         if (SHORT_ARRAYS[i] == null) {
82             Log.v("Parameter", "SHORT_ARRAYS is null at " + i);
83         }
84         if (OBJECT_ARRAYS[i] == null) {
85             Log.v("Parameter", "OBJECT_ARRAYS is null at " + i);
86         }
87     }
88 }
89
90
91 // Must call initreturnvalues and freereturnvalues afterwards.
92 public void setIndex(int index) {
93     if (index < (RANGE + 1)) {
94         this.index = index;
95     } else {
96         throw new IllegalArgumentException("Requested size too large. " +
97             index);
98     }
99 }
100
101 public int getIndex() {
102     return this.index;

```

```

103     }
104
105     public int getSize() {
106         return DEFAULTSIZE * index;
107     }
108
109
110     public Iterator<Integer> iterator() {
111         return new RangeIterator();
112     }
113
114     private class RangeIterator implements Iterator<Integer> {
115         private int index;
116
117         public RangeIterator() {
118             index = -1;
119         }
120
121         public boolean hasNext() {
122             return index < RANGE;
123         }
124
125         public Integer next() {
126             if (!hasNext()) {
127                 throw new NoSuchElementException();
128             }
129             index++;
130             setIndex(index);
131             return DEFAULTSIZE * index;
132         }
133     }
134
135     public void remove() {
136         throw new UnsupportedOperationException();
137     }
138 }
139
140 public boolean[] retrieveBooleanArray() {
141     return BOOLEAN_ARRAYS[index];
142 }
143
144 public byte[] retrieveByteArray() {
145     return BYTE_ARRAYS[index];
146 }
147
148 public char[] retrieveCharArray() {
149     return CHAR_ARRAYS[index];
150 }
151
152 public double[] retrieveDoubleArray() {
153     return DOUBLE_ARRAYS[index];
154 }
155
156 public float[] retrieveFloatArray() {
157     return FLOAT_ARRAYS[index];
158 }
159
160 public int[] retrieveIntArray() {
161     return INT_ARRAYS[index];
162 }

```

```

163
164 public long[] retrieveLongArray() {
165     return LONG_ARRAYS[index];
166 }
167
168 public short[] retrieveShortArray() {
169     return SHORT_ARRAYS[index];
170 }
171
172 public Object[] retrieveObjectArray() {
173     return OBJECT_ARRAYS[index];
174 }
175
176 public Object retrieveObject() {
177     return OBJECTS[index];
178 }
179
180 public Class retrieveClass() {
181     return OBJECTS[index].getClass();
182 }
183
184 public String retrieveString() {
185     String ret = STRINGS[index];
186     return ret;
187 }
188
189 public Throwable retrieveThrowable() {
190     return THROWABLES[index];
191 }
192
193 public ByteBuffer retrieveDirectByteBuffer() {
194     return BYTEBUFFER;
195 }
196
197 private static void generateAll() {
198     int index = RANGE - 1;
199     int size = MAXSIZE - DEFAULTSIZE;
200     generateMax();
201     while (index > -1) {
202         BOOLEAN_ARRAYS[index] = Arrays.copyOf(BOOLEAN_ARRAYS[RANGE], size);
203         CHAR_ARRAYS[index] = Arrays.copyOf(CHAR_ARRAYS[RANGE], size);
204         BYTE_ARRAYS[index] = Arrays.copyOf(BYTE_ARRAYS[RANGE], size);
205         INT_ARRAYS[index] = Arrays.copyOf(INT_ARRAYS[RANGE], size);
206         LONG_ARRAYS[index] = Arrays.copyOf(LONG_ARRAYS[RANGE], size);
207         SHORT_ARRAYS[index] = Arrays.copyOf(SHORT_ARRAYS[RANGE], size);
208         DOUBLE_ARRAYS[index] = Arrays.copyOf(DOUBLE_ARRAYS[RANGE], size);
209         FLOAT_ARRAYS[index] = Arrays.copyOf(FLOAT_ARRAYS[RANGE], size);
210
211         OBJECT_ARRAYS[index] = Arrays.copyOf(OBJECT_ARRAYS[RANGE], size);
212
213         OBJECTS[index] = OBJECT;
214         THROWABLES[index] = THROWABLE;
215         STRINGS[index] = STRING_BUILDER.substring(0, size);
216
217         index--;
218         size -= DEFAULTSIZE;
219     }
220 }
221
222 private static void generateMax() {

```



```

223     char c = 0;
224     boolean b = true;
225     byte by = 0;
226     int v = 0;
227     long l = 0;
228     short s = 0;
229     double d = 0;
230     float f = 0;
231
232     BOOLEAN_ARRAYS[RANGE] = new boolean[MAXSIZE];
233     CHAR_ARRAYS[RANGE] = new char[MAXSIZE];
234     BYTE_ARRAYS[RANGE] = new byte[MAXSIZE];
235     INT_ARRAYS[RANGE] = new int[MAXSIZE];
236     LONG_ARRAYS[RANGE] = new long[MAXSIZE];
237     SHORT_ARRAYS[RANGE] = new short[MAXSIZE];
238     DOUBLE_ARRAYS[RANGE] = new double[MAXSIZE];
239     FLOAT_ARRAYS[RANGE] = new float[MAXSIZE];
240     OBJECT_ARRAYS[RANGE] = new Object[MAXSIZE];
241
242     for (int i = 0; i < MAXSIZE; i++) {
243         STRING_BUILDER.append(c);
244
245         BOOLEAN_ARRAYS[RANGE][i] = b;
246         CHAR_ARRAYS[RANGE][i] = c;
247         BYTE_ARRAYS[RANGE][i] = by;
248         INT_ARRAYS[RANGE][i] = v;
249         LONG_ARRAYS[RANGE][i] = l;
250         SHORT_ARRAYS[RANGE][i] = s;
251         DOUBLE_ARRAYS[RANGE][i] = d;
252         FLOAT_ARRAYS[RANGE][i] = f;
253         OBJECT_ARRAYS[RANGE][i] = OBJECT;
254         OBJECTS[RANGE] = OBJECT;
255         THROWABLES[RANGE] = THROWABLE;
256
257         b = !b;
258         by = (byte) ((by + 1) % Byte.MAX_VALUE);
259         v = (v + 1) % Integer.MAX_VALUE;
260         l = (l + 1) % Long.MAX_VALUE;
261         s = (short) ((s + 1) % Short.MAX_VALUE);
262         d = (d + 0.1);
263         f = (f + 0.1f);
264         c = (char) ((char) (c + '\u0001') % (char) Character.MAX_VALUE);
265     }
266     STRINGS[RANGE] = STRING_BUILDER.substring(0, MAXSIZE);
267 }
268
269 private int index;
270 private static boolean generated = false;
271
272 private static final StringBuilder STRING_BUILDER = new StringBuilder();
273 private static final Object OBJECT = new PermissionInfo();
274 private static final Throwable THROWABLE = new Exception();
275
276 private static final String[] STRINGS = new String[RANGE + 1];
277 private static final Object[] OBJECTS = new PermissionInfo[RANGE + 1];
278 private static final Throwable[] THROWABLES = new Exception[RANGE + 1];
279
280 private static final boolean[][] BOOLEAN_ARRAYS = new boolean[RANGE + 1][];
281 private static final byte[][] BYTE_ARRAYS = new byte[RANGE + 1][];
282 private static final char[][] CHAR_ARRAYS = new char[RANGE + 1][];

```

```

283     private static final double[][] DOUBLE_ARRAYS = new double[RANGE + 1][];
284     private static final float[][] FLOAT_ARRAYS = new float[RANGE + 1][];
285     private static final int[][] INT_ARRAYS = new int[RANGE + 1][];
286     private static final long[][] LONG_ARRAYS = new long[RANGE + 1][];
287     private static final short[][] SHORT_ARRAYS = new short[RANGE + 1][];
288     private static final Object[][] OBJECT_ARRAYS = new Object[RANGE + 1][];
289
290     private static final ByteBuffer BYTEBUFFER = ByteBuffer.allocateDirect
291         (NIO_MAXSIZE);
292
293 }

```

BenchmarkRegistry.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import java.util.LinkedList;
4 import java.util.List;
5
6 import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.JavaCounterparts;
7
8 public class BenchmarkRegistry {
9
10     private static List<Benchmark> benchmarks;
11
12     public static long repetitions;
13     public static final long CHECK_INTERRUPTED_INTERVAL = 1000;
14
15     public static List<Benchmark> getBenchmarks() {
16         return benchmarks;
17     }
18
19     public static void init(long reps) throws ClassNotFoundException {
20         repetitions = reps;
21         benchmarks = new LinkedList<Benchmark>();
22         Class jCounterparts = Class.forName("fi.helsinki.cs.tituomin" +
23             ".nativebenchmark.benchmark.JavaCounterparts");
24         Class threadClass = Class.forName("java.lang.Thread");
25         initNative(reps, CHECK_INTERRUPTED_INTERVAL, jCounterparts,
26             JavaCounterparts.INSTANCE, threadClass);
27     }
28
29     public static native void initNative(long repetitions, long interval,
30         Class javaCounterparts,
31         JavaCounterparts counterInstance,
32         Class threadClass);
33
34     public static native void setRepetitions(long repetitions);
35
36     public static native void interruptNative();
37
38     public static native void resetInterruptFlag();
39 }

```

BenchmarkResult.java

```
1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.util.Log;
4
5 import java.util.HashMap;
6 import java.util.Map;
7 import java.util.Map.Entry;
8
9
10 public class BenchmarkResult {
11
12     public BenchmarkResult() {
13         values = new String[ESTIMATED_CAPACITY];
14         valueCount = 0;
15     }
16
17     public void put(String label, String value) {
18         Integer labelIndex = labelIndexes.get(label);
19         if (labelIndex == null) {
20             lastIndex++;
21             if (lastIndex > labels.length) {
22                 Log.e("BenchmarkResults", "Error, too many kinds of values, " +
23                     "increase capacity!");
24             }
25             labels[lastIndex] = label;
26             labelIndexes.put(label, lastIndex);
27             labelIndex = lastIndex;
28         }
29         values[labelIndex] = value;
30         valueCount++;
31     }
32
33     public String get(String label) {
34         Integer index = labelIndexes.get(label);
35         if (index != null) {
36             return values[index];
37         } else {
38             return null;
39         }
40     }
41
42     public String get(int i) {
43         return values[i];
44     }
45
46     public void putAll(BenchmarkResult other) {
47         String[] otherValues = other.getValues();
48         for (int i = 0; i < size(); i++) {
49             if (otherValues[i] != null) {
50                 values[i] = otherValues[i];
51                 valueCount++;
52             }
53         }
54     }
55
56     public void putAll(Map<String, String> map) {
```

```

57         for (Entry<String, String> entry : map.entrySet()) {
58             put(entry.getKey(), entry.getValue());
59         }
60     }
61
62     public String[] getValues() {
63         return values;
64     }
65
66     public static String getLabel(int i) {
67         return labels[i];
68     }
69
70     public static String[] labels() {
71         return labels;
72     }
73
74     public boolean isEmpty() {
75         return (valueCount == 0);
76     }
77
78     public static int size() {
79         return (lastIndex + 1);
80     }
81
82     private static final int ESTIMATED_CAPACITY = 200;
83     private String[] values;
84     private int valueCount;
85
86     private static String[] labels = new String[ESTIMATED_CAPACITY];
87     private static Map<String, Integer> labelIndexes = new HashMap<String,
88         Integer>(ESTIMATED_CAPACITY);
89     private static int lastIndex = -1;
90 }

```

BenchmarkRunner.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.os.SystemClock;
4 import android.util.Log;
5 import android.util.Pair;
6
7 import java.io.File;
8 import java.io.FileInputStream;
9 import java.io.FileNotFoundException;
10 import java.io.IOException;
11 import java.io.InputStream;
12 import java.io.OutputStream;
13 import java.io.PrintWriter;
14 import java.lang.reflect.Method;
15 import java.lang.reflect.Modifier;
16 import java.util.ArrayList;
17 import java.util.Arrays;
18 import java.util.Collections;

```

```

19 import java.util.Date;
20 import java.util.HashMap;
21 import java.util.Iterator;
22 import java.util.List;
23 import java.util.Map;
24 import java.util.zip.ZipEntry;
25 import java.util.zip.ZipOutputStream;
26
27 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
28 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool
29     .RunnerException;
30
31 import static fi.helsinki.cs.tituomin.nativebenchmark.Utills.colPr;
32
33 public enum BenchmarkRunner {
34     INSTANCE; // singleton enum pattern
35
36     private static final String SEPARATOR = ",";
37     private static final String MISSING_VALUE = "-";
38     private static final long WARMUP_REPS = 50000;
39     private static BenchmarkParameter benchmarkParameter;
40     private static List<MeasuringTool> measuringTools;
41     private static int benchmarkCounter = 0;
42
43     private static boolean interrupted = false;
44
45     public static BenchmarkParameter getBenchmarkParameter() {
46         if (benchmarkParameter == null) {
47             benchmarkParameter = new BenchmarkParameter();
48         }
49         return benchmarkParameter;
50     }
51
52     private BenchmarkRunner() {
53         this.allocatingRepetitions = -1;
54     }
55
56     public void initTools(ToolConfig conf, long repetitions, long
57         allocRepetitions) throws IOException, InterruptedException {
58
59         conf.setDefaultRepetitions(this.repetitions);
60         conf.setDefaultRunAllBenchmarks(this.runAllBenchmarks);
61         if (this.allocatingRepetitions > 0) {
62             conf.setDefaultAllocRepetitions(this.allocatingRepetitions);
63         }
64
65         measuringTools = new ArrayList<MeasuringTool>();
66         for (MeasuringTool tool : conf) {
67             measuringTools.add(tool);
68         }
69     }
70
71     private long repetitions;
72     private long allocatingRepetitions;
73     private CharSequence appRevision;
74     private CharSequence appChecksum;
75     private File cacheDir;
76     private boolean runAllBenchmarks;
77     private boolean runAtMaxSpeed;
78     private String benchmarkSubstring;

```

```

79
80 public enum BenchmarkSet {
81     ALLOC,
82     NON_ALLOC
83 }
84
85 ;
86 private BenchmarkSet benchmarkSet;
87
88 public BenchmarkRunner setRepetitions(long x) {
89     repetitions = x;
90     return this;
91 }
92
93 public BenchmarkRunner setAllocatingRepetitions(long x) {
94     allocatingRepetitions = x;
95     return this;
96 }
97
98 public BenchmarkRunner setAppRevision(CharSequence x) {
99     appRevision = x;
100    return this;
101 }
102
103 public BenchmarkRunner setAppChecksum(CharSequence x) {
104     appChecksum = x;
105     return this;
106 }
107
108 public BenchmarkRunner setCacheDir(File x) {
109     cacheDir = x;
110     return this;
111 }
112
113 public BenchmarkRunner setRunAllBenchmarks(boolean x) {
114     runAllBenchmarks = x;
115     return this;
116 }
117
118 public BenchmarkRunner setRunAtMaxSpeed(boolean x) {
119     runAtMaxSpeed = x;
120     return this;
121 }
122
123 public BenchmarkRunner setBenchmarkSubstring(String x) {
124     benchmarkSubstring = x;
125     return this;
126 }
127
128 public BenchmarkRunner setBenchmarkSet(BenchmarkSet x) {
129     benchmarkSet = x;
130     return this;
131 }
132
133 public void runBenchmarks(ApplicationState mainUI, ToolConfig config,
134                          File dataDir) {
135     interrupted = false;
136
137     try {
138         BenchmarkRegistry.init(this.repetitions);

```

```

139     MeasuringTool.setDataDir(dataDir);
140     Log.v(TAG, config.toString());
141     initTools(config, this.repetitions, this.allocatingRepetitions);
142 } catch (Exception e) {
143     mainUI.updateState(ApplicationState.State.ERROR);
144     Log.e("BenchmarkRunner", "Error initialising", e);
145     return;
146 }
147
148 benchmarkParameter = getBenchmarkParameter();
149 BenchmarkInitialiser.init(benchmarkParameter);
150
151 List<Benchmark> allBenchmarks = BenchmarkRegistry.getBenchmarks();
152
153 long seed = System.currentTimeMillis();
154 Log.i(TAG, String.format("Random seed %d", seed));
155 java.util.Random random = new java.util.Random(seed);
156 Collections.shuffle(allBenchmarks, random);
157 try {
158     Init.initEnvironment(true); // run warmup at max speed
159 } catch (IOException e) {
160     handleException(e, mainUI);
161     return;
162 }
163 for (MeasuringTool tool : measuringTools) {
164     if (tool == null) {
165         return;
166     }
167     if (interrupted) {
168         return;
169     }
170
171     List<Benchmark> benchmarks = new ArrayList<Benchmark>();
172
173     if (this.benchmarkSubstring == null) {
174         this.benchmarkSubstring = "";
175     }
176     String substringToApply = "";
177
178     String filter = tool.getFilter();
179     if (filter != null && !filter.equals("")) {
180         substringToApply = tool.getFilter().toLowerCase();
181     }
182     this.benchmarkSubstring = substringToApply;
183
184     for (Benchmark b : allBenchmarks) {
185         boolean selected;
186         if (tool.runAllBenchmarks()) {
187             selected = true;
188         } else if (!substringToApply.equals("")) {
189             selected = (
190                 b.getClass().getSimpleName().toLowerCase().indexOf(
191                     substringToApply) != -1);
192         } else {
193             selected = (
194                 b.representative() &&
195                 ((!b.isAllocating()) && this.benchmarkSet
196                     == BenchmarkSet.NON_ALLOC) ||
197                 (b.isAllocating() && this.benchmarkSet ==
198                     BenchmarkSet.ALLOC));

```

```

199     }
200     if (b.isNonvirtual() &&
201         b.from() != "C" &&
202         b.to() != "J") {
203         if (L.log) {
204             Log.i(TAG, String.format("skipping nonvirtual %s", b
205                                     .id()));
206         }
207         selected = false;
208     }
209     if (selected) {
210         benchmarks.add(b);
211     }
212 }
213 int numOfBenchmarks = benchmarks.size();
214
215 if (L.log) {
216     Log.i(TAG, tool.getClass().getSimpleName());
217 }
218
219 if (!tool.ignore()) {
220     // set the slower CPU frequency etc. after the warmup
221     // round(s), taking less time
222     if (!this.runAtMaxSpeed) {
223         try {
224             Init.initEnvironment(false);
225         } catch (IOException e) {
226             handleException(e, mainUI);
227             return;
228         }
229     }
230 }
231
232 int max_rounds = tool.getRounds();
233 String measurementID = Utils.getUUID();
234 File resultFile = new File(dataDir, "benchmarks-" + measurementID
235                             + ".csv");
236 long startTime = SystemClock.uptimeMillis();
237 Date start = new Date();
238 long endTime = 0;
239
240 int round = -1;
241 boolean labelsWritten = false;
242
243 ROUNDLOOP:
244 while (++round < max_rounds) {
245     benchmarkCounter = 0;
246     PrintWriter tempWriter = null;
247     File tempFile = new File(this.cacheDir, "benchmarks-temp.csv");
248     try {
249         tempWriter = Utils.makeWriter(tempFile, false);
250     } catch (FileNotFoundException e) {
251         handleException(e, mainUI);
252         return;
253     }
254
255     List<BenchmarkResult> collectedData;
256
257     try {

```



```

259         if (tool.explicitGC()) {
260             System.gc();
261             Thread.sleep(500);
262         }
263     } catch (InterruptedException e) {
264         logE("Measuring thread was interrupted");
265         mainUI.updateState(
266             ApplicationState.State.INTERRUPTED);
267         interrupted = true;
268         break ROUNDLOOP;
269     }
270     int count = benchmarks.size();
271     int j = 0;
272     for (Benchmark benchmark : benchmarks) {
273         if (L.log) {
274             Log.i(TAG, (count - j) + " left");
275             Log.i(TAG, benchmark.getClass().getSimpleName());
276         }
277         ;
278         j++;
279         try {
280             collectedData = runSeries(benchmark, mainUI, tool,
281                                     round, max_rounds, count, j);
282         } catch (RunnerException e) {
283             logE("Exception was thrown", e.getCause());
284             mainUI.updateState(
285                 ApplicationState.State.ERROR);
286             interrupted = true;
287             break ROUNDLOOP;
288         } catch (InterruptedException e) {
289             logE("Measuring thread was interrupted");
290             mainUI.updateState(
291                 ApplicationState.State.INTERRUPTED);
292
293             interrupted = true;
294             break ROUNDLOOP;
295         }
296
297         if (collectedData.isEmpty() || tool.ignore()) {
298             continue;
299         }
300
301         endTime = SystemClock.uptimeMillis();
302
303         // print data
304         for (BenchmarkResult result : collectedData) {
305             for (int i = 0; i < BenchmarkResult.size(); i++) {
306                 String value = result.get(i);
307                 if (value == null) {
308                     value = MISSING_VALUE;
309                 }
310                 tempWriter.print(value);
311                 tempWriter.print(SEPARATOR);
312             }
313             tempWriter.println("");
314             tempWriter.flush();
315         }
316     }
317     endTime = SystemClock.uptimeMillis();
318     tempWriter.close();

```

```

319
320         if (tool.ignore()) {
321             continue;
322         }
323
324         InputStream in = null;
325         OutputStream out = null;
326         PrintWriter resultWriter = null;
327         try {
328             resultWriter = Utils.makeWriter(resultFile, true);
329             // note: labels should be written last, after
330             // all possible keys have been created
331             if (!labelsWritten) {
332                 String[] labels = BenchmarkResult.labels();
333                 for (int i = 0; i < BenchmarkResult.size(); i++) {
334                     resultWriter.print(labels[i] + SEPARATOR);
335                 }
336                 resultWriter.println("");
337                 resultWriter.close();
338                 labelsWritten = true;
339             }
340
341             in = new FileInputStream(tempFile);
342             out = Utils.makeOutputStream(resultFile, true);
343             Utils.copyStream(in, out);
344         } catch (Exception e) {
345             mainUI.updateState(ApplicationState.State.ERROR);
346             Log.e("BenchmarkRunner", "Error writing results", e);
347             interrupted = true;
348             break ROUNDLOOP;
349         } finally {
350             try {
351                 if (in != null) {
352                     in.close();
353                 }
354                 if (out != null) {
355                     out.flush();
356                     out.close();
357                 }
358                 if (resultWriter != null) {
359                     resultWriter.close();
360                 }
361             } catch (IOException e) {
362                 mainUI.updateState(ApplicationState.State.ERROR);
363                 Log.e("BenchmarkRunner", "Error closing files", e);
364                 interrupted = true;
365                 break ROUNDLOOP;
366             }
367         }
368     }
369
370     if (!tool.ignore()) {
371         // there has been at least one succesful round
372         writeMeasurementMetadata(
373             new File(dataDir, "measurements.txt"),
374             measurementID,
375             (this.runAtMaxSpeed ? Init.CPUFREQ_MAX : Init.CPUFREQ),
376             startTime, endTime, start, tool, numOfBenchmarks,
377             round);
378     }

```

```

379         List<String> filenames = tool.getFileNames();
380         if (!filenames.isEmpty()) {
381             OutputStream os = null;
382
383             try {
384                 File zip = new File(dataDir, "perfdata-" +
385                     measurementID + ".zip");
386                 os = Utils.makeOutputStream(zip, false);
387                 Log.v("BenchmarkRunner", "filenames " + filenames
388                     .size());
389                 File measurementMetadataFile = new File(
390                     dataDir, "benchmarks-" + measurementID + "" +
391                     ".csv");
392                 if (measurementMetadataFile.exists()) {
393                     filenames.add(measurementMetadataFile
394                         .getAbsolutePath());
395                 }
396                 writeToZipFile(os, filenames, measurementID);
397                 deleteFiles(filenames);
398             } catch (FileNotFoundException e) {
399                 logE(e);
400             } catch (IOException e) {
401                 logE("Error writing zip file.", e);
402             }
403             try {
404                 if (os != null) {
405                     os.close();
406                 }
407             } catch (IOException e) {
408                 logE("Error closing file.", e);
409             }
410         }
411     }
412
413 }
414 mainUI.updateState(
415     ApplicationState.State.MEASURING_FINISHED);
416 }
417
418 private void writeMeasurementMetadata(
419     File catalogFile, String measurementID, int cpuFreq,
420     long startTime, long endTime, Date start, MeasuringTool tool,
421     int numOfBenchmarks, int rounds) {
422
423     Date end = new Date();
424     PrintWriter c = null;
425     try {
426         c = Utils.makeWriter(catalogFile, true);
427
428         c.println("");
429         for (Pair<String, String> pair : tool.configuration()) {
430             colPr(c, pair.first, pair.second);
431         }
432         colPr(c, "id", measurementID);
433         colPr(c, "cpu-freq", cpuFreq);
434         colPr(c, "logfile", LogAccess.filename());
435         colPr(c, "rounds", rounds);
436         colPr(c, "start", start);
437         colPr(c, "end", end);
438         colPr(c, "duration", Utils.humanTime(endTime - startTime));

```

```

439         colPr(c, "benchmarks", numOfBenchmarks);
440         colPr(c, "code-revision", this.appRevision);
441         colPr(c, "code-checksum", this.appChecksum);
442         colPr(c, "benchmark-set", this.benchmarkSet);
443         colPr(c, "substring-filter", this.benchmarkSubstring);
444         c.println("");
445     } catch (IOException e) {
446         logE(e);
447     } finally {
448         c.close();
449     }
450 }
451
452 private final static String TAG = "BenchmarkRunner";
453
454 private static void logE(String message, Throwable e) {
455     Log.e(TAG, message, e);
456 }
457
458 private static void logE(Throwable e) {
459     Log.e(TAG, "exception", e);
460 }
461
462 private static void logE(String msg) {
463     Log.e(TAG, msg);
464 }
465
466 private static void deleteFiles(List<String> filenames) {
467     for (String filename : filenames) {
468         boolean success = new File(filename).delete();
469         if (!success) {
470             logE("Error deleting file " + filename);
471         }
472     }
473 }
474
475 private static void handleException(Exception e, ApplicationState UI) {
476     logE(e);
477     UI.updateState(ApplicationState.State.ERROR);
478     return;
479 }
480
481 private static void
482 writeToZipFile(OutputStream os, List<String> filenames, String mID)
483     throws IOException {
484     ZipOutputStream zos = new ZipOutputStream(os);
485     final int byteCount = 512 * 1024;
486     byte[] bytes = new byte[byteCount];
487     for (String filename : filenames) {
488         try {
489             InputStream is = Utils.makeInputStream(filename);
490             ZipEntry entry = new ZipEntry(mID + "/" + new File(filename)
491                 .getName());
492             int bytesRead = -1;
493             zos.putNextEntry(entry);
494             while ((bytesRead = is.read(bytes, 0, byteCount)) != -1) {
495                 zos.write(bytes, 0, bytesRead);
496             }
497             zos.closeEntry();
498         } finally {

```

```

499         zos.flush();
500     }
501 }
502 try {
503     zos.close();
504 } catch (IOException e) {
505     logE(e);
506 }
507 }
508
509 private static List<BenchmarkResult> runSeries(
510     Benchmark benchmark, ApplicationState mainUI, MeasuringTool tool,
511     int roundNo, int roundCount, int benchmarkCount, int benchmarkIndex)
512     throws InterruptedException, RunnerException {
513
514     List<BenchmarkResult> compiledMetadata = new
515         ArrayList<BenchmarkResult>();
516
517     if (Thread.interrupted()) {
518         throw new InterruptedException();
519     }
520     BenchmarkParameter bPar = getBenchmarkParameter();
521     BenchmarkResult introspected;
522     try {
523         introspected = inspectBenchmark(benchmark);
524     } catch (ClassNotFoundException e) {
525         Log.e("BenchmarkRunner", "Could not find class", e);
526         return compiledMetadata;
527     }
528
529     String refTypesString = introspected.get("has_reference_types");
530     boolean hasRefTypes = (refTypesString != null) && (refTypesString
531         .equals("1"));
532
533     String parameterCountString = introspected.get("parameter_count");
534     int parameterCount = (parameterCountString == null) ? -1 : Integer
535         .parseInt(parameterCountString);
536
537     boolean dynamicParameters =
538         benchmark.dynamicParameters() ||
539         (hasRefTypes && (-1 < parameterCount &&
540             parameterCount < 2));
541
542     Iterator<Integer> iterator = bPar.iterator();
543     Integer i;
544     int j = -1;
545     if (iterator.hasNext()) {
546         i = iterator.next();
547         j++;
548     } else {
549         i = null;
550     }
551     while (i != null) {
552         if (Thread.interrupted()) {
553             throw new InterruptedException();
554         }
555         bPar.setUp(); // (I) needs tearDown (see II)
556
557         try {
558             tool.startMeasuring(benchmark);

```

```

559         benchmarkCounter++;
560     } catch (IOException e) {
561         logE("Measuring caused IO exception", e);
562         if (mainUI.userWantsToRetry(e)) {
563             continue; // without incrementing i
564         } else {
565             throw new InterruptedException("User wants to abort");
566         }
567     } finally {
568         bPar.tearDown(); // (II) needs setUp (see I)
569     }
570     if (tool.explicitGC() && benchmarkCounter % 25 == 0) {
571         System.gc();
572         Thread.sleep(350);
573     }
574
575     String message = String.format(
576         "%s%s round %d/%d benchmark %d/%d range %d/%d",
577         tool.getClass().getSimpleName(),
578         tool.isWarmupRound() ? " (warmup)" : "",
579         roundNo + 1,
580         roundCount,
581         benchmarkIndex,
582         benchmarkCount,
583         j,
584         dynamicParameters ? BenchmarkParameter.RANGE : 1
585     );
586     mainUI.updateState(ApplicationState.State.MILESTONE, message);
587
588     List<BenchmarkResult> measurements = tool.getMeasurements();
589     for (BenchmarkResult measurement : measurements) {
590         if (!measurement.isEmpty()) {
591             if (dynamicParameters) {
592                 measurement.put("dynamic_size", "" + i);
593             } else {
594                 measurement.put("dynamic_size", MISSING_VALUE);
595             }
596             measurement.put("id", benchmark.id());
597             measurement.put("class", benchmark.getClass()
598                 .getSimpleName());
599             measurement.putAll(introspected);
600             compiledMetadata.add(measurement);
601         }
602     }
603     // if parameter size can be varied, vary it - else break with
604     // first size
605     if (!dynamicParameters) {
606         break;
607     }
608     if (iterator.hasNext()) {
609         i = iterator.next();
610         j++;
611     } else {
612         break;
613     }
614 }
615 return compiledMetadata;
616 }
617
618

```

```

619 private static BenchmarkResult inspectBenchmark(Benchmark benchmark)
620     throws ClassNotFoundException {
621     BenchmarkResult bdata = new BenchmarkResult();
622     int seqNo = benchmark.sequenceNo();
623     String from = benchmark.from();
624     String to = benchmark.to();
625     bdata.put("no", "" + benchmark.sequenceNo());
626     bdata.put("description", benchmark.description());
627     bdata.put("from", from);
628     bdata.put("to", to);
629     bdata.put("representative", benchmark.representative() ? "1" : "0");
630     bdata.put("nonvirtual", benchmark.isNonvirtual() ? "1" : "0");
631
632     if (seqNo == -1) {
633         bdata.put("custom", "1");
634         return bdata;
635     }
636
637     Class c = Class.forName(
638         String.format(
639             "fi.helsinki.cs.tituomin.nativebenchmark.benchmark" +
640             ".J2CBenchmark%05d",
641             seqNo));
642
643     Method[] methods = c.getDeclaredMethods();
644     for (int i = 0; i < methods.length; i++) {
645
646         Method m = methods[i];
647         int modifiers = m.getModifiers();
648
649         if (Modifier.isNative(modifiers)) {
650             return inspectMethod(m, bdata);
651         }
652     }
653     return bdata;
654 }
655
656 private static BenchmarkResult inspectMethod(Method m, BenchmarkResult
657     bdata) {
658
659     Class[] parameters = m.getParameterTypes();
660     List<Class> parameterList = new ArrayList<Class>(Arrays.asList
661         (parameters));
662     int modifiers = m.getModifiers();
663
664     Map<String, Integer> parameterTypes = new HashMap<String, Integer>();
665     for (Class param : parameterList) {
666         Integer previousValue = null;
667         String param_typename = param.getSimpleName();
668         previousValue = parameterTypes.get(param_typename);
669         parameterTypes.put(
670             param_typename,
671             (previousValue == null ? 1 : ((int) previousValue) + 1));
672     }
673     for (String typename : parameterTypes.keySet()) {
674         bdata.put("parameter_type_" + typename + "_count", parameterTypes
675             .get(typename) + "");
676     }
677
678     Class returnType = m.getReturnType();

```

```

679
680     boolean hasRefTypes = false;
681     parameterList.add(returnType);
682     for (Class cl : parameterList) {
683         if (Object.class.isAssignableFrom(cl)) {
684             hasRefTypes = true;
685             break;
686         }
687     }
688
689     bdata.put("has_reference_types", hasRefTypes ? "1" : "0");
690     bdata.put("parameter_type_count", parameterTypes.keySet().size() + "");
691     bdata.put("parameter_count", parameters.length + "");
692     bdata.put("return_type", returnType.getCanonicalName());
693     bdata.put("native_static", Modifier.isStatic(modifiers) ? "1" : "0");
694     bdata.put("native_private", Modifier.isPrivate(modifiers) ? "1" : "0");
695     bdata.put("native_protected", Modifier.isProtected(modifiers) ? "1" :
696         "0");
697     bdata.put("native_public", Modifier.isPublic(modifiers) ? "1" : "0");
698
699     return bdata;
700 }
701 }

```

BenchmarkSelector.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkInitialiser;
4 //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
5 //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRunner;
6
7 import android.app.Activity;
8 import android.app.AlertDialog;
9 import android.app.Dialog;
10 import android.app.DialogFragment;
11 import android.app.Notification;
12 import android.app.NotificationManager;
13 import android.app.PendingIntent;
14 import android.app.TaskStackBuilder;
15 import android.content.Context;
16 import android.content.DialogInterface;
17 import android.content.Intent;
18 import android.content.res.Resources;
19 import android.media.RingtoneManager;
20 import android.net.Uri;
21 import android.os.Bundle;
22 import android.os.Environment;
23 import android.util.Log;
24 import android.view.View;
25 import android.view.WindowManager;
26 import android.widget.AdapterView;
27 import android.widget.AdapterView.OnItemClickListener;
28 import android.widget.ArrayAdapter;
29 import android.widget.Button;

```



```

30 import android.widget.Checkable;
31 import android.widget.NumberPicker;
32 import android.widget.Spinner;
33 import android.widget.TextView;
34
35 import java.io.File;
36 import java.io.InputStream;
37 import java.io.OutputStream;
38 import java.util.Map;
39
40 public class BenchmarkSelector extends Activity {
41     /**
42      * Called when the activity is first created.
43      */
44     @Override
45     public void onCreate(Bundle savedInstanceState) {
46         super.onCreate(savedInstanceState);
47         this.requestWindowFeature(getWindow().FEATURE_NO_TITLE);
48         setContentView(R.layout.main);
49
50         this.retry = false;
51
52         this.resultView = (TextView) findViewById(R.id.resultview);
53         this.button = (Button) findViewById(R.id.mybutton);
54         this.repView = (TextView) findViewById(R.id.repetitions);
55         this.numPick = (NumberPicker) findViewById(R.id.picker_num);
56         this.expPick = (NumberPicker) findViewById(R.id.picker_exp);
57
58         NumberPicker.OnValueChangeListener listener = new RepsListener();
59
60         numPick.setMinValue(1);
61         numPick.setMaxValue(9);
62         numPick.setValue(1);
63         expPick.setMinValue(0);
64         expPick.setMaxValue(9);
65         expPick.setValue(6);
66         numPick.setOnValueChangedListener(listener);
67         expPick.setOnValueChangedListener(listener);
68
69         listener.onValueChange(numPick, 0, 0);
70
71         if (getResources().getString(R.string.app_dirty).equals("1")) {
72             this.resultView.setText(R.string.warning_changed);
73         }
74
75         final Resources resources = getResources();
76         this.runner = BenchmarkRunner.INSTANCE
77             .setAppChecksum(resources.getText(R.string.app_checksum))
78             .setAppRevision(resources.getText(R.string.app_revision))
79             .setCacheDir(getCacheDir());
80
81         File sd = Environment.getExternalStorageDirectory();
82         dataDir = new File(sd, "results");
83         dataDir.mkdir();
84
85         this.controller = new BenchmarkController(this, dataDir);
86
87         this.socketCommunicator = new SocketCommunicator();
88
89         File configFile = new File(

```

```

90         Environment.getExternalStorageDirectory(),
91         "nativebenchmark_setup.json"
92     );
93     try {
94         if (!configFile.exists()) {
95             InputStream templateStream = getResources()
96                 .openRawResource(R.raw.setup);
97             OutputStream configFileStream = Utils.makeOutputStream
98                 (configFile, false);
99             Utils.copyStream(templateStream, configFileStream);
100         }
101         this.configuration = ToolConfig.readConfigFile();
102     } catch (Exception e) {
103         String msg = getResources().getString(R.string.config_error);
104         Log.e(TAG, msg, e);
105         displayMessage(ApplicationState.State.INIT_FAIL, msg);
106     }
107
108     if (this.configuration != null) {
109         initSpinner(this.configuration);
110         if (allocationArray == null) {
111             allocationArray = new byte[1024 * 1024 * 100];
112         }
113     }
114     this.socketCommunicator.startServer(this.controller, this.runner);
115 }
116
117 public void onDestroy() {
118     socketCommunicator.stopServer();
119     super.onDestroy();
120 }
121
122 private void initSpinner(Map<String, ToolConfig> conf) {
123     Spinner spinner = (Spinner) findViewById(R.id.config_spinner);
124     String keys[] = conf.keySet().toArray(new String[1]);
125     ArrayAdapter<CharSequence> adapter = new ArrayAdapter(this, android.R
126         .layout.simple_spinner_item, keys);
127
128     int indexOfDefault = -1;
129     while (++indexOfDefault < keys.length &&
130         !keys[indexOfDefault].equals("default")) ;
131
132     adapter.setDropDownViewResource(android.R.layout
133         .simple_spinner_dropdown_item);
134     spinner.setAdapter(adapter);
135     spinner.setOnItemClickListener(new OnItemSelectedListener() {
136         public void onItemClick(
137             AdapterView<?> parent, View view,
138             int pos, long id) {
139             selectedConfiguration = (String) parent.getItemAtPosition(pos);
140         }
141
142         public void onNothingSelected(AdapterView<?> parent) {
143         }
144     });
145
146     spinner.setSelection(indexOfDefault);
147
148 }
149

```

```

150     public void displayMessage(ApplicationState.State state, String message) {
151         displayMessage(state.stringId, message);
152     }
153
154     public void displayMessage(int id) {
155         this.resultView.setText(id);
156     }
157
158     public void displayMessage(String message) {
159         this.resultView.setText(message);
160     }
161
162     public void displayMessage(int id, String message) {
163         this.resultView.setText(getResources().getString(id) + " " + message);
164     }
165
166     private class StateChanger implements Runnable {
167         public void run() {
168             ApplicationState.DetailedState detailedState = controller
169                 .getState();
170             ApplicationState.State state = detailedState.state;
171             String message = detailedState.message;
172
173             if (message == null) {
174                 displayMessage(state.stringId);
175             } else {
176                 displayMessage(state.stringId, message);
177             }
178             switch (state) {
179                 case MEASURING_STARTED:
180                     getWindow().addFlags(WindowManager.LayoutParams
181                         .FLAG_KEEP_SCREEN_ON);
182                     expPick.setEnabled(false);
183                     numPick.setEnabled(false);
184                     switchButton(button);
185                     state = ApplicationState.State.MILESTONE;
186                     break;
187                 case MILESTONE:
188                     break;
189                 case ERROR:
190                     displayMessage(ApplicationState.State.ERROR, message);
191                     // intended fallthrough
192                 case INTERRUPTED:
193                     // intended fallthrough
194                 case MEASURING_FINISHED:
195                     getWindow().clearFlags(WindowManager.LayoutParams
196                         .FLAG_KEEP_SCREEN_ON);
197                     stateThread.interrupt();
198                     notifyFinished();
199                     // intended fallthrough
200                 case INITIALED:
201                     resetButton(button);
202                     numPick.setEnabled(true);
203                     expPick.setEnabled(true);
204                     break;
205                 case INIT_FAIL:
206             }
207         }
208     }
209

```

```

210 public boolean userWantsToRetry(final Exception e) {
211     runOnUiThread(new Runnable() {
212         public void run() {
213             DialogFragment dialog = new RetryDialog(e.getMessage());
214             dialog.show(getFragmentManager(), "foo");
215         }
216     });
217     boolean waiting = true;
218     while (waiting) {
219         synchronized (this) {
220             try {
221                 this.wait();
222                 waiting = false;
223             } catch (InterruptedException ie) {
224                 waiting = true;
225             }
226         }
227     }
228     return this.retry;
229 }
230
231 private void setRetry(boolean answer) {
232     this.retry = answer;
233     synchronized (this) {
234         this.notify();
235     }
236 }
237
238 private void resetButton(Button button) {
239     button.setText(R.string.start_task);
240     button.setOnClickListener(new View.OnClickListener() {
241         public void onClick(View v) {
242             startMeasuring(v);
243         }
244     });
245 }
246
247 private void switchButton(Button bt) {
248     bt.setText(R.string.end_task);
249     bt.setOnClickListener(
250         new View.OnClickListener() {
251             public void onClick(View v) {
252                 button.setText(R.string.end_task_confirmation);
253                 button.setOnClickListener(
254                     new View.OnClickListener() {
255                         public void onClick(View v) {
256                             displayMessage(ApplicationState.State
257                                 .INTERRUPTING.stringId);
258                             controller.interruptMeasuring();
259                         }
260                     });
261             }
262         });
263 }
264
265 private boolean isChecked(int id) {
266     return ((Checkable) findViewById(id)).isChecked();
267 }
268
269

```

```

270     private String textValue(int id) {
271         return ((TextView) findViewById(id)).getText().toString();
272     }
273
274     public void startMeasuring(View view) {
275         this.runner
276             .setRepetitions(repetitions)
277             .setAllocatingRepetitions(Long.parseLong(textValue(R.id
278                 .alloc_reps)))
279             .setBenchmarkSubstring(textValue(R.id.benchmark_substring)
280                 .toLowerCase())
281             .setRunAllBenchmarks(isChecked(R.id.checkbox_long))
282             .setRunAtMaxSpeed(isChecked(R.id.checkbox_max))
283             .setBenchmarkSet(isChecked(R.id.run_alloc) ?
284                 BenchmarkRunner.BenchmarkSet.ALLOC :
285                 BenchmarkRunner.BenchmarkSet.NON_ALLOC);
286
287         stateThread = new Thread(
288             new Runnable() {
289                 public void run() {
290                     while (!Thread.currentThread().isInterrupted()) {
291                         runOnUiThread(new StateChanger());
292                         try {
293                             Thread.sleep(5000);
294                         } catch (InterruptedException e) {
295                             break;
296                         }
297                     }
298                 }
299             });
300
301         stateThread.start();
302         allocationArray = null;
303         controller.startMeasuring(this.runner, this.configuration.get
304             (selectedConfiguration));
305     }
306
307     private void notifyFinished() {
308         Uri alarmSound = RingtoneManager.getDefaultUri(
309             RingtoneManager.TYPE_NOTIFICATION);
310
311         Notification.Builder mBuilder =
312             new Notification.Builder(this)
313                 .setSmallIcon(android.R.drawable.ic_media_play)
314                 .setContentTitle(getResources().getText(R.string
315                     .measuring_finished))
316                 .setContentText(getResources().getText(R.string
317                     .measuring_finished))
318                 .setSound(alarmSound);
319
320         Intent resultIntent = new Intent(this, BenchmarkSelector.class);
321
322         TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
323         stackBuilder.addParentStack(BenchmarkSelector.class);
324         stackBuilder.addNextIntent(resultIntent);
325         PendingIntent resultPendingIntent =
326             stackBuilder.getPendingIntent(
327                 0,
328                 PendingIntent.FLAG_UPDATE_CURRENT
329             );

```

```

330     NotificationManager mNotificationManager =
331         (NotificationManager) getSystemService(Context
332             .NOTIFICATION_SERVICE);
333     // mId allows you to update the notification later on.
334     mNotificationManager.notify(1, mBuilder.build());
335 }
336
337 private class RepsListener implements NumberPicker.OnValueChangeListener {
338     public void onValueChange(NumberPicker picker, int oldVal, int newVal) {
339         long exp = BenchmarkSelector.this.expPick.getValue();
340         long value = BenchmarkSelector.this.numPick.getValue();
341         while (exp-- > 0) {
342             value *= 10;
343         }
344         repetitions = value;
345         repView.setText("" + repetitions);
346     }
347 }
348
349 private class RetryDialog extends DialogFragment {
350     private String message;
351
352     public RetryDialog(String message) {
353         this.message = message;
354     }
355
356     @Override
357     public Dialog onCreateDialog(Bundle savedInstanceState) {
358         // Use the Builder class for convenient dialog construction
359         AlertDialog.Builder builder = new AlertDialog.Builder(getActivity
360             ());
361         builder.setMessage(getResources().getText(R.string
362             .retry_question) + ":\n" + this.message)
363             .setPositiveButton(R.string.retry_answer_positive, new
364                 DialogInterface.OnClickListener() {
365                     public void onClick(DialogInterface dialog,
366                         int id) {
367                         setRetry(true);
368                     }
369                 })
370             .setNegativeButton(R.string.retry_answer_negative, new
371                 DialogInterface.OnClickListener() {
372                     public void onClick(DialogInterface dialog,
373                         int id) {
374                         setRetry(false);
375                     }
376                 });
377         // Create the AlertDialog object and return it
378         return builder.create();
379     }
380 }
381
382 private long repetitions;
383 private boolean retry;
384 private TextView textView, resultView, repView;
385 private NumberPicker numPick, expPick;
386 private Button button;
387 private Thread stateThread;
388 private static byte[] allocationArray;
389 private File dataDir;

```

```

390     private BenchmarkController controller;
391     private SocketCommunicator socketCommunicator;
392     private BenchmarkRunner runner;
393
394     private String selectedConfiguration;
395
396     private static final String TAG = "BenchmarkSelector";
397
398     private Map<String, ToolConfig> configuration;
399
400     static {
401         System.loadLibrary("nativebenchmark");
402     }
403
404 }

```

Init.java

```

1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import java.io.IOException;
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Init {
8
9      private static final String TAG = "NativeBenchmark";
10     public static final int CPUFREQ = 400000;
11     public static final int CPUFREQ_MAX = 1000000;
12
13     public static List<String> initScript(int cpufreq) {
14         List<String> script = new ArrayList<String>();
15         script.add(
16             "echo \"userspace\" > " +
17                 "/sys/devices/system/cpu/cpu0/cpufreq" +
18                 "/scaling_governor");
19         script.add(
20             "echo \"" + cpufreq + "\" > " +
21                 "/sys/devices/system/cpu/cpu0/cpufreq" +
22                 "/scaling_setspeed");
23         return script;
24     }
25
26
27     public static void initEnvironment(boolean maxSpeed) throws IOException {
28         ShellEnvironment.runAsRoot(initScript(maxSpeed ? CPUFREQ_MAX :
29             CPUFREQ));
30     }
31 }

```

L.java

```
1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 public class L {
4     // Set L.log to false to statically remove
5     // debugging logging statements from
6     // measuring code.
7     public static final boolean og = false;
8 }
```

LogAccess.java

```
1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.util.Log;
4
5 import java.io.File;
6 import java.io.IOException;
7 import java.util.regex.Pattern;
8
9 public class LogAccess {
10
11     private static final String LOGCAT_COMMAND =
12         "logcat -f %s " +
13         "-b main -b system -b radio -b events " +
14         "-v time";
15
16     public static void start(File dir) throws IOException {
17         currentRunId = Utils.getUUID();
18         mark(START, currentRunId);
19         String filename = new File(dir, filename()).getAbsolutePath();
20         String command = String.format(LOGCAT_COMMAND, filename);
21         logcatProcess = Runtime.getRuntime().exec(command);
22     }
23
24     public static void end() {
25         if (logcatProcess != null) {
26             mark(END, currentRunId);
27             logcatProcess.destroy();
28             logcatProcess = null;
29         }
30     }
31
32     private static void mark(String type, String id) {
33         Log.println(LOGLEVEL, TAG, marker(type) + id);
34     }
35
36     private static String marker(String type) {
37         return "[" + type + "] ";
38     }
39
40     public static String filename() {
```



```

41     return "log-" + currentRunId + ".txt";
42 }
43
44 private static Pattern makeMarkerPattern(String type) {
45     return Pattern.compile("[-.: [0-9]]+ I/" + TAG + "\\([ [0-9]+\\): \\["
46         + type + "\\] " + currentRunId);
47 }
48
49 private static final int LOGLEVEL = Log.INFO;
50 private static final String TAG = "LogAccess";
51 private static final String START = "Start";
52 private static final String END = "End";
53 private static String currentRunId;
54 private static Process logcatProcess;
55 }

```

measuringtool/AllocatingBenchmarkLongRunningWrapper.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
7
8 public class AllocatingBenchmarkLongRunningWrapper extends
9     AllocatingBenchmarkWrapper {
10
11     public AllocatingBenchmarkLongRunningWrapper(Benchmark b, long r) {
12         super(b, r);
13     }
14
15     private static final long MAX_REPS = Long.MAX_VALUE;
16
17     public void begin(MeasuringTool tool) throws InterruptedException,
18         IOException {
19         Benchmark benchmark = getBenchmark();
20         init(benchmark);
21         tool.putMeasurement("repetitions", this.repetitions + "");
22         tool.start(this);
23         tool.finishMeasurement();
24     }
25
26     public void run() {
27         // This method ensures the garbage collector is run
28         // every benchmark.maxrepetitions iteration
29         // but otherwise the measurement is
30         // run for a period long enough for profiling.
31         Benchmark benchmark = getBenchmark();
32         long interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL;
33         long division, remainder;
34         long repetitions = MAX_REPS;
35
36         division = repetitions / interval + 1;
37         remainder = repetitions % interval + 1;

```

```

38
39     while (--division != 0) {
40         interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL + 1;
41         while (--interval != 0) {
42             try {
43                 benchmark.run();
44                 System.gc();
45                 Thread.sleep(GC_PAUSE_MS);
46             } catch (InterruptedException e) {
47                 setInterrupted();
48                 return;
49             } catch (Exception e) {
50                 setException(e);
51                 return;
52             }
53         }
54         if (Thread.currentThread().isInterrupted()) {
55             setInterrupted();
56             return;
57         }
58     }
59
60     while (--remainder != 0) {
61         try {
62             benchmark.run();
63             System.gc();
64             Thread.sleep(GC_PAUSE_MS);
65         } catch (InterruptedException e) {
66             setInterrupted();
67             return;
68         } catch (Exception e) {
69             setException(e);
70             return;
71         }
72     }
73
74 }
75
76 }

```

measuringtool/AllocatingBenchmarkShortRunningWrapper.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6
7 public class AllocatingBenchmarkShortRunningWrapper extends
8     AllocatingBenchmarkWrapper {
9
10     public AllocatingBenchmarkShortRunningWrapper(Benchmark b, long r) {
11         super(b, r);
12     }
13

```

```

14     private static final long MULTIPLIER = 25;
15
16     public void begin(MeasuringTool tool) throws InterruptedException,
17         IOException {
18         Benchmark benchmark = getBenchmark();
19         init(benchmark);
20         long reps = MULTIPLIER;
21         reps += 1;
22         while (--reps != 0) {
23             tool.putMeasurement("repetitions", this.repetitions + "");
24             tool.putMeasurement("multiplier", MULTIPLIER + "");
25             try {
26                 tool.start(getBenchmark());
27                 tool.finishMeasurement();
28                 System.gc();
29                 Thread.sleep(GC_PAUSE_MS);
30             } catch (InterruptedException e) {
31                 setInterrupted();
32                 return;
33             } catch (Exception e) {
34                 setException(e);
35                 return;
36             }
37         }
38     }
39 }

```

measuringtool/AllocatingBenchmarkWrapper.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
4
5 public class AllocatingBenchmarkWrapper extends RunningWrapper {
6
7     public AllocatingBenchmarkWrapper(Benchmark b, long repetitions) {
8         super(b);
9         this.repetitions = repetitions;
10    }
11
12    public static final int GC_PAUSE_MS = 400;
13
14    public void init(Benchmark benchmark) {
15        super.init(benchmark);
16        benchmark.setRepetitions(repetitions);
17    }
18
19    protected long repetitions;
20
21 }

```

```
1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.util.Pair;
4
5 public class BasicOption implements MeasuringOption {
6
7     public BasicOption(OptionSpec type) {
8         this(type, null);
9     }
10
11     public BasicOption(OptionSpec type, String value) {
12         this.type = type;
13         this.value = value;
14     }
15
16     // -----
17
18     public OptionSpec id() {
19         return this.type;
20     }
21
22     public void set(String value) {
23         this.value = value;
24     }
25
26     public OptionSpec type() {
27         return this.type;
28     }
29
30     public String value() {
31         return value;
32     }
33
34     public Pair<String, String> toStringPair() {
35         return new Pair<String, String>(this.type.id(), this.value);
36     }
37
38     public String toString() {
39         return this.type() + " " + this.value();
40     }
41
42     // ----
43
44     private OptionSpec type;
45     private String value;
46
47     // ----
48
49 }
```

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.util.Log;
4
5 import java.io.IOException;
6 import java.text.DateFormat;
7 import java.util.ArrayList;
8 import java.util.Date;
9 import java.util.LinkedList;
10 import java.util.List;
11 import java.util.Random;
12
13 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
14 import fi.helsinki.cs.tituomin.nativebenchmark.ShellEnvironment;
15
16
17 public abstract class CommandlineTool extends MeasuringTool {
18
19     private static final long REPETITIONS = Long.MAX_VALUE;
20
21     public CommandlineTool(int rounds, long repetitions, long allocreps,
22                             boolean warmup, boolean x) throws IOException,
23                             InterruptedException {
24         super(rounds, REPETITIONS, allocreps, false, x);
25         this.fileNames = new ArrayList<String>();
26     }
27
28     protected abstract String command();
29
30     protected String formatParameter(MeasuringOption option) {
31         throw new UnsupportedOperationException();
32     }
33
34     protected String formatDefaultParameter(MeasuringOption option) {
35         if (option.type() == OptionSpec.COMMAND_STRING) {
36             return option.value();
37         } else {
38             return formatParameter(option);
39         }
40     }
41
42     public void initCommand() {
43         List<String> commandline = new LinkedList<String>();
44         commandline.addAll(generateCommandLineArguments());
45         this.command = "";
46         for (String s : commandline) {
47             this.command += s + " ";
48         }
49     }
50
51     protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
52         options = super.specifyAllowedOptions(options);
53         options.add(OptionSpec.COMMAND_STRING);
54         return options;
55     }
56
57     protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
58                                                     options) {
59         options.add(new BasicOption(OptionSpec.COMMAND_STRING, command()));
60         return options;
61     }
62

```

```

61     }
62
63     protected void init() throws IOException, InterruptedException {
64         super.init();
65         if (!ShellEnvironment.runAsRoot(initScript())) {
66             throw new IOException("Error executing as root.");
67         }
68     }
69
70     protected abstract List<String> initScript();
71
72     public boolean isLongRunning() {
73         return true;
74     }
75
76     public void start(Runnable benchmark)
77         throws InterruptedException, IOException {
78         if (Thread.interrupted()) {
79             throw new InterruptedException();
80         }
81
82         initCommand();
83         BenchmarkRegistry.resetInterruptFlag();
84         Thread benchmarkThread = new Thread(benchmark);
85         Random r = new Random();
86         int delay = r.nextInt(20);
87
88         benchmarkThread.start();
89         Thread.sleep(delay);
90
91         try {
92             ShellEnvironment.run(this.command);
93         } catch (InterruptedException e) {
94             throw e;
95         } finally {
96             benchmarkThread.interrupt();
97             BenchmarkRegistry.interruptNative();
98             benchmarkThread.join();
99         }
100     }
101
102     public void setFilename(String name, String path) {
103         filenames.add(path + "/" + name);
104         putMeasurement("Filename", name);
105     }
106
107     public List<String> getFilenames() {
108         return filenames;
109     }
110
111
112     public void setUUID(String uuid) {
113         putMeasurement("UUID", uuid);
114     }
115
116     public List<String> generateCommandLineArguments() {
117         List<String> result = new LinkedList<String>();
118         for (OptionSpec type : allowedOptions) {
119             MeasuringOption option = options.get(type);
120

```

```

121         if (option == null) {
122             Log.v("mt", type + " is null");
123         } else {
124             result.add(formatDefaultParameter(option));
125         }
126     }
127     return result;
128 }
129
130 private Date startDate;
131 private Date endDate;
132 private long startTime;
133 private String command;
134
135 private List<String> filenames;
136 private static final DateFormat dateFormat = DateFormat
137     .getDateTimeInstance();
138
139 }

```

measuringtool/JavaSystemNanoResponseTimeRecorder.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 public class JavaSystemNanoResponseTimeRecorder extends ResponseTimeRecorder {
6
7     public JavaSystemNanoResponseTimeRecorder(
8         int i,
9         long reps,
10        long allocreps,
11        boolean warmup,
12        boolean runAllBenchmarks
13    ) throws IOException, InterruptedException {
14        super(i, reps, allocreps, warmup, runAllBenchmarks);
15    }
16
17    public void start(Runnable benchmark)
18        throws InterruptedException, IOException {
19        long endTime, startTime;
20        startTime = System.nanoTime();
21        benchmark.run();
22        endTime = System.nanoTime();
23        String delta = "" + (endTime - startTime);
24        putMeasurement("response_time", delta);
25        putMeasurement("time_unit", "nanoseconds");
26    }
27 }

```

```
1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.util.LinkedList;
6 import java.util.List;
7
8 import fi.helsinki.cs.tituomin.nativebenchmark.Utills;
9
10 public class LinuxPerfRecordTool extends CommandLineTool {
11
12     public LinuxPerfRecordTool
13         (
14             int rounds,
15             long repetitions,
16             long allocreps,
17             boolean warmup,
18             boolean runAllBenchmarks) throws
19             IOException, InterruptedException {
20         super(rounds, repetitions, allocreps, warmup, runAllBenchmarks);
21     }
22
23     protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
24         options = super.specifyAllowedOptions(options);
25         options.add(OptionSpec.OUTPUT_FILEPATH);
26         options.add(OptionSpec.MEASURE_LENGTH); // must be last
27         return options;
28     }
29
30     protected List<String> initScript() {
31         List<String> commands = new LinkedList<String>();
32         commands.add("echo \"0\" > /proc/sys/kernel/kptr_restrict");
33         commands.add("echo \"-1\" > /proc/sys/kernel/perf_event_paranoid");
34         return commands;
35     }
36
37     protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
38                                                     options) {
39         options = super.defaultOptions(options);
40         String uuid = Utills.getUUID();
41         String filename = generateFilename(uuid);
42         String basePath = getPerfDir().getPath() + "/";
43         setFilename(filename, basePath);
44         setUUID(uuid);
45         options.add(new BasicOption(OptionSpec.OUTPUT_FILEPATH, basePath +
46                                     filename));
47         return options;
48     }
49
50     protected void init() throws IOException, InterruptedException {
51         super.init();
52         getPerfDir().mkdir();
53     }
54
55     private File getPerfDir() {
56         return new File(getDataDir(), "perf");
57     }
58 }
```



```

57     }
58
59     protected String command() {
60         return "perf record -a -g";
61     }
62
63     private String generateFilename(String uuid) {
64         return "perf-" + uuid + ".data";
65     }
66
67     public String formatParameter(MeasuringOption option) {
68         if (option.type() == OptionSpec.OUTPUT_FILEPATH) {
69             return "--output=" + option.value();
70         } else if (option.type() == OptionSpec.MEASURE_LENGTH) {
71             return "sleep " + option.value();
72         }
73         return super.formatParameter(option);
74     }
75 }

```

measuringtool/MeasuringOption.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.util.Pair;
4
5 public interface MeasuringOption {
6
7     public void set(String value);
8
9     public OptionSpec id();
10
11     public String value();
12
13     public OptionSpec type();
14
15     public Pair<String, String> toStringPair();
16 }

```

measuringtool/MeasuringTool.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3
4 import android.util.Log;
5 import android.util.Pair;
6
7 import java.io.File;
8 import java.io.IOException;

```

```

9 import java.text.SimpleDateFormat;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import java.util.HashMap;
13 import java.util.HashSet;
14 import java.util.LinkedList;
15 import java.util.List;
16 import java.util.Locale;
17 import java.util.Map;
18 import java.util.Set;
19
20 import fi.helsinki.cs.tituomin.nativebenchmark.ApplicationState;
21 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
22 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
23 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkResult;
24
25 public abstract class MeasuringTool implements Runnable {
26
27     public MeasuringTool
28         (
29             int rounds,
30             long repetitions,
31             long allocRepetitions,
32             boolean warmup,
33             boolean runAllBenchmarks
34         ) throws
35             IOException, InterruptedException {
36         clearMeasurements();
37         specifyOptions();
38         this.rounds = rounds;
39         this.repetitions = repetitions;
40         this.allocRepetitions = allocRepetitions;
41         this.warmup = warmup;
42         this.explicitGC = !warmup;
43         this.runAllBenchmarks = runAllBenchmarks;
44         init();
45     }
46
47     public static synchronized void userInterrupt() {
48         userInterrupted = true;
49         BenchmarkRegistry.interruptNative();
50     }
51
52     public static synchronized boolean userInterrupted() {
53         if (userInterrupted == true) {
54             userInterrupted = false;
55             return true;
56         }
57         return false;
58     }
59
60     protected void init() throws IOException, InterruptedException {
61     }
62
63     protected abstract void start(Runnable benchmark)
64         throws InterruptedException, IOException;
65
66     public boolean isLongRunning() {
67         return false;
68     }

```

```

69
70 public RunningWrapper wrap(Benchmark benchmark) {
71     if (!benchmark.isAllocating()) {
72         // Default running algorithm
73         return new RunningWrapper(benchmark);
74     } else {
75         // the benchmark does allocations, we have
76         // to limit the amount of loops -> compensate
77         // by repeating the loop many times
78         if (isLongRunning()) {
79             return new AllocatingBenchmarkLongRunningWrapper(benchmark,
80                 allocRepetitions);
81         } else {
82             return new AllocatingBenchmarkShortRunningWrapper(benchmark,
83                 allocRepetitions);
84         }
85     }
86 }
87
88 public void startMeasuring(Benchmark benchmark) throws
89     InterruptedException, IOException, RunnerException {
90     String benchmarkName = benchmark.getClass().getSimpleName();
91     clearMeasurements();
92     setDefaultOptions();
93     benchmark.setRepetitions(this.repetitions);
94     RunningWrapper wrapper = wrap(benchmark);
95     Date start = null, end = null;
96
97     start = new Date();
98     wrapper.begin(this);
99     end = new Date();
100
101     putMeasurement("start", DATEFORMAT.format(start));
102     putMeasurement("end", DATEFORMAT.format(end));
103
104     if (wrapper.wasInterrupted() && userInterrupted()) {
105         throw new InterruptedException("Interrupted by user");
106     }
107     if (wrapper.exceptionWasThrown()) {
108         throw new RunnerException(wrapper.getException());
109     }
110 }
111
112 public class RunnerException extends Exception {
113     public RunnerException(Throwable t) {
114         super(t);
115     }
116 }
117
118
119 public void run() {
120     try {
121         start(benchmark);
122     } catch (InterruptedException e) {
123         Log.e("BenchmarkRunner", "Measuring was interrupted ", e);
124     } catch (IOException e) {
125         Log.e("BenchmarkRunner", "IOException", e);
126     }
127 }
128

```

```

129     public boolean explicitGC() {
130         return this.explicitGC;
131     }
132
133     public boolean runAllBenchmarks() {
134         return this.runAllBenchmarks;
135     }
136
137     public void setExplicitGC(boolean e) {
138         this.explicitGC = e;
139     }
140
141     protected abstract List<MeasuringOption>
142     defaultOptions(List<MeasuringOption> container);
143
144     protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec>
145                                                     container) {
146         return container;
147     }
148
149     protected void specifyOptions() {
150         this.allowedOptions = specifyAllowedOptions(
151             new LinkedList<OptionSpec>());
152
153         if (this.requiredOptions == null) {
154             this.requiredOptions = new HashSet<OptionSpec>();
155         }
156         for (OptionSpec o : this.allowedOptions) {
157             if (o.required()) {
158                 this.requiredOptions.add(o);
159             }
160         }
161     }
162
163     protected void setDefaultOptions() {
164         for (MeasuringOption op : defaultOptions(
165             new LinkedList<MeasuringOption>())) {
166             setOption(op);
167         }
168         ;
169     }
170
171     public MeasuringTool set(String id, String value) {
172         setOption(new BasicOption(OptionSpec.byId(id), value));
173         return this;
174     }
175
176     public MeasuringTool set(OptionSpec spec, String value) {
177         setOption(new BasicOption(spec, value));
178         return this;
179     }
180
181     public String getOption(OptionSpec id) {
182         return this.options.get(id).value();
183     }
184
185     public void setDescription(String d) {
186         this.description = d;
187     }
188

```

```

189 public void setOption(MeasuringOption option) {
190     if (this.allowedOptions == null) {
191         specifyOptions();
192     }
193     if (!allowedOptions.contains(option.type())) {
194         throw new UnsupportedOperationException();
195     } else {
196         this.options =
197             options != null ?
198                 options :
199                 new HashMap<OptionSpec, MeasuringOption>();
200
201         this.options.put(option.type(), option);
202     }
203 }
204
205 public void setBenchmark(Benchmark b) {
206     benchmark = b;
207 }
208
209 public boolean ignore() {
210     return false;
211 }
212
213 private boolean hasRequiredOptions() {
214     return options.keySet().containsAll(requiredOptions);
215 }
216
217 protected List<OptionSpec> allowedOptions;
218 protected Set<OptionSpec> requiredOptions;
219 protected Map<OptionSpec, MeasuringOption> options;
220
221 private List<ApplicationState> observers;
222
223 private BenchmarkResult currentMeasurement;
224 private List<BenchmarkResult> measurements;
225
226 protected long repetitions;
227
228 protected void putMeasurement(String key, String value) {
229     currentMeasurement.put(key, value);
230 }
231
232 protected void finishMeasurement() {
233     currentMeasurement = new BenchmarkResult();
234     measurements.add(currentMeasurement);
235 }
236
237 public List<BenchmarkResult> getMeasurements() {
238     for (BenchmarkResult measurement : measurements) {
239         if (this.options != null) {
240             for (MeasuringOption op : options.values()) {
241                 measurement.put(
242                     op.toStringPair().first,
243                     op.toStringPair().second);
244             }
245         }
246     }
247     return this.measurements;
248 }

```

```

249
250 public void clearMeasurements() {
251     this.currentMeasurement = new BenchmarkResult();
252     this.measurements = new LinkedList<BenchmarkResult>();
253     measurements.add(currentMeasurement);
254 }
255
256 private static final List<String> emptyList = new ArrayList<String>();
257
258 public List<String> get_filenames() {
259     return emptyList;
260 }
261
262 public int getRounds() {
263     return rounds;
264 }
265
266 public static void setDataDir(File dir) {
267     dataDir = dir;
268 }
269
270 public static File getDataDir() {
271     return dataDir;
272 }
273
274 public void setFilter(String substring) {
275     filterSubstring = substring;
276 }
277
278 public String getFilter() {
279     return filterSubstring;
280 }
281
282 public List<Pair<String, String>> configuration() {
283     List<Pair<String, String>> pairs = new ArrayList<Pair<String,
284         String>>();
285     pairs.add(new Pair<String, String>("tool", this.getClass()
286         .getSimpleName()));
287     pairs.add(new Pair<String, String>("repetitions", "" + this
288         .repetitions));
289     pairs.add(new Pair<String, String>("description", this.description));
290     pairs.add(new Pair<String, String>("warmup", "" + this.warmup));
291     if (options != null) {
292         for (MeasuringOption opt : options.values()) {
293             pairs.add(new Pair<String, String>(opt.type().id(), opt.value
294                 ()));
295         }
296     }
297     return pairs;
298 }
299
300 public boolean isWarmupRound() {
301     return warmup;
302 }
303
304 private static File dataDir;
305
306 private Benchmark benchmark;
307 private int rounds;
308 private long allocRepetitions;

```

```

309     private String description;
310     private String filterSubstring;
311     protected boolean warmup;
312     private boolean explicitGC;
313     private boolean runAllBenchmarks;
314     private static boolean userInterrupted = false;
315     private final static String TAG = "MeasuringTool";
316
317     public static class UnsupportedOperationException extends RuntimeException {
318     }
319
320     private static SimpleDateFormat DATEFORMAT = new SimpleDateFormat("MM-dd " +
321         "HH:mm:ss.SSS", Locale.US);
322
323 }

```

measuringtool/MockCommandlineTool.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 public class MockCommandlineTool extends CommandlineTool {
7
8     public MockCommandlineTool(int i, long reps) throws IOException,
9         InterruptedException {
10         super(i, reps, reps, false, false);
11     }
12
13     protected List<String> initScript() {
14         return null;
15     }
16
17     public boolean ignore() {
18         return true;
19     }
20
21     protected String command() {
22         return "cat /dev/null";
23     }
24
25 }

```

measuringtool/OptionSpec.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.util.HashMap;

```

```

4 import java.util.Map;
5
6 public enum OptionSpec {
7
8     COMMAND_STRING("Command run", "command_string", true),
9     OUTPUT_FILEPATH("Output path", "output_filepath", true),
10    MEASURE_LENGTH("Measuring time (sec)", "measure_length", true),
11    VARIABLE("Variable parameter in benchmark", "variable", false),
12    CPUFREQ("Fixed CPU frequency", "cpu_freq", true);
13
14    OptionSpec(String name, String id, boolean required) {
15        this.name = name;
16        this.id = id;
17        this.required = required;
18        put(id, this);
19    }
20
21    public String id() {
22        return id;
23    }
24
25    public boolean required() {
26        return required;
27    }
28
29    public static OptionSpec byId(String id) {
30        return specs.get(id);
31    }
32
33    private static void put(String id, OptionSpec value) {
34        getSpecs().put(id, value);
35    }
36
37    public static Map<String, OptionSpec> getSpecs() {
38        if (specs == null) {
39            specs = new HashMap<String, OptionSpec>();
40        }
41        return specs;
42    }
43
44    private String name;
45    private String id;
46    private boolean required;
47
48    private static Map<String, OptionSpec> specs;
49
50 }

```

measuringtool/PlainRunner.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4 import java.util.List;
5

```



```

6 public class PlainRunner extends MeasuringTool {
7
8     public PlainRunner(int i, long reps, long allocreps) throws IOException,
9         InterruptedException {
10         super(i, reps, allocreps, true, false);
11     }
12
13     protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
14         options) {
15         return options;
16     }
17
18     protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
19         options = super.specifyAllowedOptions(options);
20         options.add(OptionSpec.CPUFREQ);
21         return options;
22     }
23
24     public boolean explicitGC() {
25         return false;
26     }
27
28     public boolean ignore() {
29         return true;
30     }
31
32     public long repetitions() {
33         return 10000;
34     }
35
36     public void start(Runnable benchmark)
37         throws InterruptedException, IOException {
38         benchmark.run();
39     }
40 }

```

measuringtool/ResponseTimeRecorder.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.os.SystemClock;
4
5 import java.io.IOException;
6 import java.util.List;
7
8 public class ResponseTimeRecorder extends MeasuringTool {
9
10     public ResponseTimeRecorder(
11         int rounds,
12         long reps,
13         long allocreps,
14         boolean warmup,
15         boolean x)
16         throws IOException, InterruptedException {
17         super(rounds, reps, allocreps, warmup, x);

```

```

18     }
19
20     protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
21                                                    options) {
22         // no options needed, time is returned as is (not in extra file)
23         return options;
24     }
25
26     public boolean ignore() {
27         return warmup;
28     }
29
30     public void start(Runnable benchmark)
31         throws InterruptedException, IOException {
32         long endTime, startTime;
33         startTime = SystemClock.uptimeMillis();
34         benchmark.run();
35         endTime = SystemClock.uptimeMillis();
36         String delta = "" + (endTime - startTime);
37         putMeasurement("response_time", delta);
38         putMeasurement("time_unit", "milliseconds");
39     }
40
41 }

```

measuringtool/RunningWrapper.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6
7 public class RunningWrapper implements Runnable {
8     private Benchmark benchmark;
9     private Exception exceptionThrown;
10    private boolean interrupted;
11    private long repetitions;
12
13    public RunningWrapper(Benchmark b) {
14        benchmark = b;
15    }
16
17    protected void setException(Exception e) {
18        exceptionThrown = e;
19    }
20
21    protected void setInterrupted() {
22        interrupted = true;
23    }
24
25    public boolean exceptionWasThrown() {
26        return exceptionThrown != null;
27    }
28

```

```

29     public void setRepetitions(long r) {
30         repetitions = r;
31     }
32
33     public Benchmark getBenchmark() {
34         return benchmark;
35     }
36
37     public long getRepetitions() {
38         return repetitions;
39     }
40
41     public boolean wasInterrupted() {
42         return interrupted;
43     }
44
45     public Exception getException() {
46         return exceptionThrown;
47     }
48
49     public void init(Benchmark benchmark) {
50         interrupted = false;
51         exceptionThrown = null;
52     }
53
54     public void run() {
55         benchmark.run();
56     }
57
58     public void begin(MeasuringTool tool) throws InterruptedException,
59         IOException {
60         init(benchmark);
61         tool.start(this);
62         if (Thread.currentThread().isInterrupted()) {
63             setInterrupted();
64         }
65     }
66
67 }

```

MockObject.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 public class MockObject {
4
5     public boolean jbooleanField;
6     public byte jbyteField;
7     public char jcharField;
8     public double jdoubleField;
9     public float jfloatField;
10    public int jintField;
11    public long jlongField;
12    public short jshortField;
13    public Object jobjectField;

```

```

14
15     public static boolean jbooleanStaticField;
16     public static byte jbyteStaticField;
17     public static char jcharStaticField;
18     public static double jdoubleStaticField;
19     public static float jfloatStaticField;
20     public static int jintStaticField;
21     public static long jlongStaticField;
22     public static short jshortStaticField;
23     public static Object jobjectStaticField;
24
25     public MockObject() {
26         jbooleanField = true;
27         jbyteField = 1;
28         jcharField = 2;
29         jdoubleField = 1.2;
30         jfloatField = 3.1f;
31         jintField = 3;
32         jlongField = 4;
33         jshortField = 5;
34         jobjectField = this;
35
36         jbooleanStaticField = true;
37         jbyteStaticField = 1;
38         jcharStaticField = 2;
39         jdoubleStaticField = 1.2;
40         jfloatStaticField = 3.1f;
41         jintStaticField = 3;
42         jlongStaticField = 4;
43         jshortStaticField = 5;
44         jobjectStaticField = this;
45     }
46 }

```

ShellEnvironment.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.util.Log;
4
5 import java.io.BufferedReader;
6 import java.io.DataOutputStream;
7 import java.io.IOException;
8 import java.io.InputStream;
9 import java.io.InputStreamReader;
10 import java.util.List;
11
12 public class ShellEnvironment {
13
14     // Thank you http://muzikant-android.blogspot
15     // .fi/2011/02/how-to-get-root-access-and-execute.html
16     public static boolean runAsRoot(List<String> commands) throws IOException {
17         if (commands == null) {
18             return true;
19         }

```

```

20     boolean retval = false;
21     DataOutputStream os = null;
22     Process suProcess = null;
23     try {
24         if (null != commands && commands.size() > 0) {
25             suProcess = Runtime.getRuntime().exec("su");
26
27             os = new DataOutputStream(suProcess.getOutputStream());
28
29             // Execute commands that require root access
30             for (String currCommand : commands) {
31                 os.writeBytes(currCommand + "\n");
32                 os.flush();
33             }
34
35             os.writeBytes("exit\n");
36             os.flush();
37
38             try {
39                 int suProcessRetVal = suProcess.waitFor();
40                 if (255 != suProcessRetVal) {
41                     // Root access granted
42                     retval = true;
43                 } else {
44                     // Root access denied
45                     retval = false;
46                 }
47             } catch (Exception ex) {
48                 Log.e("ROOT", "Error executing root action", ex);
49             }
50         }
51     } catch (IOException ex) {
52         Log.w("ROOT", "Can't get root access", ex);
53     } catch (SecurityException ex) {
54         Log.w("ROOT", "Can't get root access", ex);
55     } catch (Exception ex) {
56         Log.w("ROOT", "Error executing internal operation", ex);
57     } finally {
58         if (suProcess != null) {
59             suProcess.destroy();
60         }
61     }
62     if (os != null) {
63         os.close();
64     }
65
66     return retval;
67 }
68
69 public static void run(String command)
70     throws IOException, InterruptedException {
71
72     Process process = null;
73     InputStream err = null;
74     try {
75         process = Runtime.getRuntime().exec(command);
76         err = process.getErrorStream();
77         process.waitFor();
78
79         if (process.exitValue() != 0) {

```

```

80         String line;
81         BufferedReader br = new BufferedReader(new InputStreamReader
82             (err));
83         StringBuilder sb = new StringBuilder("Command failed.\n");
84         while ((line = br.readLine()) != null) {
85             Log.e("tm", line);
86             sb.append(line);
87             sb.append("\n");
88         }
89         process.destroy();
90         br.close();
91         throw new IOException(sb.toString());
92     }
93     catch (IOException e) {
94         throw e;
95     } catch (InterruptedException e) {
96         throw e;
97     } finally {
98         if (err != null) err.close();
99         if (process != null) process.destroy();
100     }
101 }
102
103 }

```

SocketCommunicator.java

```

1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3
4  import android.util.Log;
5
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.io.OutputStream;
9  import java.io.PrintWriter;
10 import java.net.InetSocketAddress;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13 import java.net.SocketTimeoutException;
14 import java.util.Map;
15
16 public class SocketCommunicator implements ApplicationStateListener {
17     /**
18      * Thread to initialize Socket connection
19      */
20     private final Runnable InitializeConnection = new Thread() {
21         @Override
22         public void run() {
23             // initialize server socket
24             while (!Thread.currentThread().isInterrupted()) {
25                 try {
26                     server = new ServerSocket();
27                     server.setReuseAddress(true);
28                     server.bind(new InetSocketAddress(38300));

```

```

29
30 //attempt to accept a connection
31 client = server.accept();
32
33 SocketCommunicator.this.out = client.getOutputStream();
34 SocketCommunicator.this.printWriter = new PrintWriter
35     (out, true);
36 SocketCommunicator.nis = client.getInputStream();
37 try {
38     SocketCommunicator.this.output(helpMessage);
39
40     byte[] bytes = new byte[1024];
41     int numRead = 0;
42     while (numRead >= 0) {
43         SocketCommunicator.this.output("[Awaiting input]");
44         numRead = SocketCommunicator.nis.read(bytes, 0,
45             1024);
46         if (numRead < 1) {
47             executeCommand("quit");
48             return;
49         }
50         receivedCommand = new String(bytes, 0, numRead)
51             .trim();
52         executeCommand(receivedCommand);
53     }
54 } catch (IOException ioException) {
55     Log.e(SocketCommunicator.TAG, "" + ioException);
56 }
57 } catch (SocketTimeoutException e) {
58     receivedCommand = "Connection has timed out! Please try " +
59         "again";
60     Log.v(TAG, receivedCommand);
61 } catch (IOException e) {
62     Log.e(SocketCommunicator.TAG, "" + e);
63 }
64
65 if (client != null) {
66     receivedCommand = "Connection was successful!";
67     try {
68         server.close();
69     } catch (IOException e) {
70         Log.e(TAG, "Error closing server connection.");
71     } finally {
72         server = null;
73     }
74     Log.v(TAG, receivedCommand);
75 }
76 }
77 }
78 };
79
80 private class StateChanger implements Runnable {
81     public void run() {
82     }
83 }
84
85 private void executeStart(String command) {
86     String[] split = command.split(":");
87     if (split.length < 2) {
88         Log.e(TAG, "No configuration key provided.");

```

```

89         return;
90     }
91     String configKey = split[1];
92     Map<String, ToolConfig> configurations = null;
93     ToolConfig config = null;
94     try {
95         configurations = ToolConfig.readConfigFile();
96         config = configurations.get(configKey);
97     } catch (Exception e) {
98         Log.e(TAG, "Error reading configuration file.", e);
99     }
100    if (config == null) {
101        ApplicationState.DetailedState ds = new ApplicationState
102            .DetailedState(controller);
103        ds.state = ApplicationState.State.ERROR;
104        ds.message = "Could not find configuration for " + configKey;
105        stateUpdated(ds);
106        return;
107    }
108    stateThread = new Thread(
109        new Runnable() {
110            public void run() {
111                while (!Thread.currentThread().isInterrupted()) {
112                    ApplicationState.DetailedState detailedState =
113                        controller.getState();
114                    ApplicationState.State state = detailedState.state;
115                    stateUpdated(detailedState);
116                    if (state == ApplicationState.State
117                        .MEASURING_FINISHED ||
118                        state == ApplicationState.State.ERROR) {
119                        return;
120                    }
121                    try {
122                        Thread.sleep(10000);
123                    } catch (InterruptedException e) {
124                        break;
125                    }
126                }
127            }
128        }
129    );
130
131    stateThread.start();
132    this.controller.startMeasuring(this.runner, config);
133 }
134
135 private void output(String message) {
136     synchronized (this) {
137         this.printWriter.println(message);
138     }
139 }
140
141 public void executeCommand(String command) {
142     boolean executed = false;
143     command = command.trim();
144     if (command.length() == 0) {
145         return;
146     } else if (command.startsWith("start")) {
147         this.executeStart(command);
148         executed = true;

```



```

149     } else if (command.startsWith("end")) {
150         this.controller.interruptMeasuring();
151         executed = true;
152     } else if (command.startsWith("quit")) {
153         this.restartServer();
154         executed = true;
155     } else {
156         this.output(String.format("Unkown command %s", command));
157         Log.v(TAG, "" + command + " == unknown command.");
158     }
159     if (!executed) {
160         return;
161     }
162     this.output(String.format(
163         "[Executed %s]", command));
164 }
165
166 public void startServer(
167     BenchmarkController controller,
168     BenchmarkRunner runner)
169
170 {
171     this.controller = controller;
172     this.runner = runner;
173     //initialize server socket in a new separate thread
174     this.serverThread = new Thread(InitializeConnection);
175     this.serverThread.start();
176     String msg = "Attempting to connect";
177     Log.v(TAG, msg);
178 }
179
180 public boolean stopServer() {
181     try {
182         this.output("Stopping socket server.");
183         serverThread.interrupt();
184         if (SocketCommunicator.nis != null) {
185             SocketCommunicator.nis.close();
186         }
187         if (this.out != null) {
188             this.out.close();
189         }
190         if (server != null) {
191             server.close();
192         }
193         return true;
194     } catch (IOException ec) {
195         Log.e(SocketCommunicator.TAG, "Cannot close server socket" + ec);
196         return false;
197     }
198 }
199
200 public void restartServer() {
201     if (this.stopServer()) {
202         this.startServer(this.controller, this.runner);
203     }
204 }
205
206 public void stateUpdated(ApplicationState.DetailedState state) {
207     this.output(state.toString());
208 }

```

```

209
210     public static final String TAG = "SocketCommunicator";
211     public static final int TIMEOUT = 10;
212     private ServerSocket server = null;
213     private Socket client = null;
214     private OutputStream out;
215     private PrintWriter printWriter;
216     private String receivedCommand = null;
217     public static InputStream nis = null;
218     private BenchmarkController controller;
219     private Map<String, ToolConfig> configurations;
220     private BenchmarkRunner runner;
221     private Thread serverThread;
222     private Thread stateThread;
223
224     private final String helpMessage = "\n" +
225         "Measuring application ready.\n" +
226         "Available commands:\n" +
227         "  start :CONFIG_KEY\n" +
228         "    Starts measuring with the configuration\n" +
229         "    loaded from nativebenchmark_setup.json file\n" +
230         "    under the top level key CONFIG_KEY.\n" +
231         "  end\n" +
232         "    Interrupts measuring.\n";
233
234 }

```

ToolConfig.java

```

1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import android.os.Environment;
4  import android.util.Log;
5
6  import org.json.JSONArray;
7  import org.json.JSONException;
8  import org.json.JSONObject;
9
10 import java.io.File;
11 import java.io.IOException;
12 import java.lang.reflect.Constructor;
13 import java.lang.reflect.InvocationTargetException;
14 import java.util.HashMap;
15 import java.util.Iterator;
16 import java.util.Map;
17 import java.util.NoSuchElementException;
18
19 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
20
21 public class ToolConfig implements Iterable<MeasuringTool> {
22
23     public static Map<String, ToolConfig> readConfigurations(String jsonConfig)
24         throws JSONException {
25         JSONObject cfgObject = new JSONObject(jsonConfig);
26         Map<String, ToolConfig> result = new HashMap<String, ToolConfig>();

```

```

27     Iterator i = cfgObject.keys();
28     while (i.hasNext()) {
29         String key = (String) i.next();
30         result.put(key, new ToolConfig(cfgObject.getJSONObject(key)));
31     }
32     return result;
33 }
34
35 public static Map<String, ToolConfig> readConfigFile() throws
36     IOException, JSONException {
37     String jsonContents = null;
38     File configFile = new File(
39         Environment.getExternalStorageDirectory(),
40         "nativebenchmark_setup.json"
41     );
42     jsonContents = Utils.readFileToString(configFile);
43     return ToolConfig.readConfigurations(jsonContents);
44 }
45
46 public ToolConfig(JSONObject job) {
47     this.globalOptions = job;
48     this.defaultAllocRepetitions = 400;
49 }
50
51 public String toString() {
52     return this.globalOptions.toString();
53 }
54
55 public Iterator<MeasuringTool> iterator() {
56     try {
57         return new ToolIterator();
58     } catch (JSONException e) {
59         Log.e("ToolConfig", "Error reading json config", e);
60     }
61     return null;
62 }
63
64 private class ToolIterator implements Iterator<MeasuringTool> {
65     private JSONArray toolArray;
66     private int currentToolIndex;
67
68     public ToolIterator() throws JSONException {
69         currentToolIndex = -1;
70         toolArray = globalOptions.getJSONArray("tools");
71     }
72
73     public boolean hasNext() {
74         return currentToolIndex + 1 < toolArray.length();
75     }
76
77     public MeasuringTool next() {
78         MeasuringTool tool = null;
79         try {
80             tool = createTool(toolArray.getJSONObject(++currentToolIndex));
81             if (tool == null) {
82                 throw new NoSuchElementException();
83             }
84         } catch (JSONException e) {
85             Log.e("ToolConfig", "Error reading json config", e);
86         }

```

```

87         return tool;
88     }
89
90     public void remove() {
91         throw new UnsupportedOperationException();
92     }
93 }
94
95 private MeasuringTool createTool(JSONObject toolOptions) {
96
97     MeasuringTool tool = null;
98     try {
99         long repetitions = toolOptions.optLong(
100             "repetitions", globalOptions.optLong(
101                 toolOptions.optString("repetitions"), this
102                     .defaultRepetitions));
103
104         int defaultRounds = 1;
105
106         int rounds = toolOptions.optInt(
107             "rounds", globalOptions.optInt(
108                 toolOptions.optString("rounds"), defaultRounds));
109
110         long allocRepetitions = toolOptions.optLong(
111             "alloc_repetitions", globalOptions.optLong(
112                 toolOptions.optString("alloc_repetitions"),
113                 this.defaultAllocRepetitions));
114
115         boolean warmup = toolOptions.optBoolean("warmup", false);
116
117         boolean runAllBenchmarks = toolOptions.optBoolean(
118             "run_all", this.defaultRunAllBenchmarks);
119
120         Class<?> _class = Class.forName(TOOL_PACKAGE + "." + toolOptions
121             .getString("class"));
122
123         Constructor<?> ctor = _class.getConstructor(
124             Integer.TYPE, Long.TYPE, Long.TYPE, Boolean.TYPE, Boolean
125                 .TYPE);
126
127         Log.v("ToolConfig", "Tool instantiation " + rounds + " " +
128             repetitions + " " + warmup);
129
130         try {
131             tool = (MeasuringTool) ctor.newInstance(
132                 rounds, repetitions, allocRepetitions,
133                 warmup, runAllBenchmarks);
134         } catch (InvocationTargetException e) {
135             Log.e("ToolConfig", "Constructor exception", e.getCause());
136         }
137         tool.setDescription(toolOptions.optString("description", ""));
138         tool.setFilter(toolOptions.optString("filter", ""));
139         tool.setExplicitGC(toolOptions.optBoolean("gc", !warmup));
140
141         JSONObject options = toolOptions.optJSONObject("options");
142         if (options != null) {
143             Iterator keys = options.keys();
144             while (keys.hasNext()) {
145                 String key = (String) keys.next();
146                 tool.set(key, options.getString(key));

```

```

147         }
148     }
149
150     } catch (Exception e) {
151         Log.e("ToolConfig", "Error instantiating tool", e);
152         return null;
153     }
154     return tool;
155 }
156
157 public ToolConfig setDefaultRepetitions(long r) {
158     defaultRepetitions = r;
159     return this;
160 }
161
162 public ToolConfig setDefaultAllocRepetitions(long r) {
163     defaultAllocRepetitions = r;
164     return this;
165 }
166
167 public ToolConfig setDefaultRunAllBenchmarks(boolean r) {
168     defaultRunAllBenchmarks = r;
169     return this;
170 }
171
172 public BenchmarkRunner.BenchmarkSet getBenchmarkSet() {
173     String key = globalOptions.optString("benchmark_set", "non_alloc");
174     if (key.equals("alloc")) {
175         return BenchmarkRunner.BenchmarkSet.ALLOC;
176     }
177     return BenchmarkRunner.BenchmarkSet.NON_ALLOC;
178 }
179
180 private long defaultRepetitions;
181 private long defaultAllocRepetitions;
182 private boolean defaultRunAllBenchmarks;
183
184 private JSONObject globalOptions;
185 private static final String TOOL_PACKAGE = "fi.helsinki.cs.tituomin" +
186     ".nativebenchmark.measuringtool";
187 private static final String TAG = "nativebenchmark.ToolConfig";
188
189 }

```

Utils.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import java.io.BufferedInputStream;
4 import java.io.BufferedOutputStream;
5 import java.io.File;
6 import java.io.FileInputStream;
7 import java.io.FileNotFoundException;
8 import java.io.FileOutputStream;
9 import java.io.IOException;

```

```

10 import java.io.InputStream;
11 import java.io.OutputStream;
12 import java.io.PrintWriter;
13 import java.nio.MappedByteBuffer;
14 import java.nio.channels.FileChannel;
15 import java.nio.charset.Charset;
16 import java.util.UUID;
17
18
19 public class Utils {
20
21     public static String getUUID() {
22         return UUID.randomUUID().toString();
23     }
24
25     public static String humanTime(long millis) {
26         String time;
27         long seconds = millis / 1000;
28         long minutes = seconds / 60;
29         long hours = minutes / 60;
30         long seconds_total = seconds;
31         seconds %= 60;
32         minutes %= 60;
33         return (hours + "h " +
34             minutes + "m " +
35             seconds + "s " +
36             "(" + seconds_total + " s tot.)");
37     }
38
39     public static String colFmt(String label, String value) {
40         return String.format("%20s: %s", label, value);
41     }
42
43     public static void colPr(PrintWriter p, String label, Object value) {
44         p.println(colFmt(label, value.toString()));
45     }
46
47     public static void copyStream(InputStream in, OutputStream out) throws
48         IOException {
49         int count;
50         while ((count = in.read(buffer, 0, BUFFERSIZE)) != -1) {
51             out.write(buffer, 0, count);
52         }
53         out.flush();
54     }
55
56     public static PrintWriter
57     makeWriter(File dir, String filename, boolean append)
58         throws FileNotFoundException {
59         return new PrintWriter(makeOutputStream(dir, filename, append));
60     }
61
62     public static PrintWriter
63     makeWriter(File file, boolean append)
64         throws FileNotFoundException {
65         return new PrintWriter(makeOutputStream(file, append));
66     }
67
68     public static OutputStream
69     makeOutputStream(File dir, String filename, boolean append)

```

```

70         throws FileNotFoundException {
71     return makeOutputStream(new File(dir, filename), append);
72 }
73
74 public static OutputStream
75 makeOutputStream(File file, boolean append)
76     throws FileNotFoundException {
77     return new BufferedOutputStream(
78         new FileOutputStream(
79             file, append));
80 }
81
82 public static InputStream
83 makeInputStream(File file)
84     throws FileNotFoundException {
85     return new BufferedInputStream(new FileInputStream(file));
86 }
87
88
89 public static InputStream
90 makeInputStream(String filename)
91     throws FileNotFoundException {
92     return makeInputStream(new File(filename));
93 }
94
95 public static String readFileToString(File file) throws IOException {
96     FileInputStream stream = new FileInputStream(file);
97     try {
98         FileChannel fc = stream.getChannel();
99         MappedByteBuffer bb = fc.map(FileChannel.MapMode.READ_ONLY, 0, fc
100             .size());
101         return Charset.defaultCharset().decode(bb).toString();
102     } finally {
103         stream.close();
104     }
105 }
106
107 private static final int BUFFERSIZE = 128 * 1024;
108 private static byte[] buffer = new byte[BUFFERSIZE];
109 }

```

Python-komponentit (koodingenerointi)

Hakemistossa script/.

benchmark_generator.py

```
1 from templates import java_benchmark
2 from templates import java_counterparts
3 from templates import c_module
4 from templates import c_jni_function
5 from templates import c_nativemethod
6
7 from templating import put
8
9 import itertools
10 import logging
11
12 import jni_types
13 from jni_types import types, primitive_types, return_types
14
15 packagename = [
16     'fi',
17     'helsinki',
18     'cs',
19     'tituomin',
20     'nativebenchmark',
21     'benchmark']
22 library_name = 'nativebenchmark'
23 java_counterpart_classname = 'JavaCounterparts'
24 native_method_name = 'nativemethod'
25 class_counter = 0
26
27 def next_sequence_no():
28     global class_counter
29     class_counter += 1
30     return str(class_counter).zfill(5)
31
32
33 def benchmark_classname(prefix, number):
34     return prefix + 'Benchmark' + number
35
36
37 def parameter_initialisation(language, typespec, name):
38     if typespec.get('is-array', False):
39         if language == 'java':
40             expression = 'benchmarkParameter.retrieve{_type}Array()'.format(
41                 _type=typespec['java-element-type'].capitalize())
42         else:
43             expression = '{_type}ArrayValue'.format(
44                 _type=typespec['c-element-type'])
45     elif typespec.get('is-object', False):
46         if language == 'java':
47             expression = 'benchmarkParameter.retrieve{_type}()'.format(
48                 _type=typespec['java'])
49         else:
50             expression = 'benchmarkParameter.retrieve{_type}()'
51     return expression
```



```

51         expression = '{_type}Value'.format(
52             _type=typespec['c'])
53
54     elif language == 'java':
55         if typespec.get('java-literal'):
56             expression = typespec['java-literal']
57         else:
58             expression = ''
59
60     elif language == 'c':
61         if typespec.get('c-literal'):
62             expression = typespec['c-literal']
63         else:
64             expression = ''
65
66     if name:
67         return name + " = " + expression
68     else:
69         return expression
70
71
72 def modifier_combinations():
73     privacy = ['public', 'private', 'protected']
74     static = ['static', '']
75     return list(itertools.product(privacy, static))
76
77
78 def method_combinations():
79     combinations = []
80     combinations.append({
81         'description': 'no arguments or return types',
82         'representative': True,
83         'return_types': [return_types['v']],
84         'target_modifiers': modifier_combinations(),
85         'types': []})
86
87     for symbol, _type in types.iteritems():
88         combinations.append({
89             'description': 'varying count {0}'.format(symbol),
90             'representative': _type.get('representative', False),
91             'return_types': [return_types['v']],
92             'target_modifiers': [('public', '')],
93             'types': jni_types.type_combinations(
94                 size=20,
95                 typeset=[types[symbol]]})
96
97     # Start from 1 to avoid duplicates.
98     combinations.append({
99         'description': 'vary number of types',
100         'representative': True,
101         'skip': 1,
102         'return_types': [return_types['v']],
103         'target_modifiers': [('public', '')],
104         'types': jni_types.type_combinations(
105             typeset=types.values())
106 })
107
108 combinations.append({
109     'description': 'modifier combinations',
110     'representative': True,

```

```

111         'return_types': [return_types['l']],
112         'target_modifiers': modifier_combinations(),
113         'include_nonvirtual': True,
114         'types': [types['i']]
115     })
116
117     filtered_return_types = filter(
118         lambda x: x['symbol'] != 'l',
119         jni_types.type_combinations(typeset=types.values()))
120
121     combinations.append({
122         'description': 'return types',
123         'return_types': filtered_return_types,
124         'target_modifiers': [('public', '')],
125         'types': [types['i']]
126     })
127
128     return combinations
129
130
131 def generate_benchmarks():
132     global class_counter
133     java = []
134     java_callees = {}
135     c_implementations = []
136     c_runners = []
137     c_methodid_inits = []
138
139     for spec in method_combinations():
140         virtualities = [True]
141         if spec.get('include_nonvirtual'):
142             virtualities.append(False)
143         for virtualcall in virtualities:
144             for target_modifier in spec['target_modifiers']:
145                 for return_type in spec['return_types']:
146
147                     if 'representative' not in spec:
148                         representative = return_type.get(
149                             'representative', False)
150                     else:
151                         representative = spec['representative']
152
153                     representative = "true" if representative else "false"
154
155                     type_combination = spec['types']
156
157                     native_method_modifiers = " ".join(target_modifier)
158                     return_expression = parameter_initialisation(
159                         'c', return_type, None)
160
161                     # Declare/initialize the parameters that are passed
162                     # to the called function/method.
163
164                     parameter_names = []
165
166                     parameter_declarations = []
167                     parameter_initialisations = []
168                     c_parameter_declarations = []
169                     c_parameter_initialisations = []
170

```

```

171     if target_modifier[1] == 'static':
172         c_parameter_declarations.append('jclass cls')
173     else:
174         c_parameter_declarations.append('jobject instance')
175
176     for i, type_data in enumerate(type_combination):
177         parameter_names.append(
178             type_data['symbol'] + str(i + 1))
179
180         parameter_declarations.append(
181             type_data['java'] + ' ' + parameter_names[-1])
182         c_parameter_declarations.append(
183             type_data['c'] + ' ' + parameter_names[-1])
184         parameter_initialisations.append(
185             parameter_initialisation(
186                 'java', type_data, parameter_names[-1])){
187         c_parameter_initialisations.append(
188             parameter_initialisation(
189                 'c', type_data, parameter_names[-1]))
190
191     skip = spec.get('skip', 0)
192     upper_bound = len(type_combination){
193     if upper_bound == 0:
194         upper_bound = 1
195     for i in range(skip, upper_bound):
196
197         sequence_no = next_sequence_no()
198
199         for from_lang, to_lang in [
200             ('J', 'C'), ('J', 'J'),
201             ('C', 'C'), ('C', 'J')]:
202
203             pairing = (from_lang, to_lang)
204
205             if virtualcall == False:
206                 # Nonvirtual calls only apply to C -> J
207                 # instance methods J2C is generated as a proxy
208                 # to get benchmark metadata
209                 if (target_modifier[1] == 'static' or
210                     pairing in [('J', 'J'), ('C', 'C')]):
211                     continue
212
213             # 1. Set up call targets.
214
215             if to_lang == 'C':
216                 counterpart_method_name = 'nativemethod'
217             if to_lang == 'J':
218                 counterpart_method_unqualified = (
219                     'benchmark' + sequence_no)
220
221             if pairing == ('J', 'J'):
222                 if target_modifier[0] == 'private':
223                     # A private method cannot
224                     # be called from Java.
225                     # Don't generate a benchmark.
226                     continue
227
228                 if target_modifier[1] == 'static':
229                     clsname = java_counterpart_classname
230                 else:

```

```

231         clsname = 'counterpartInstance'
232         counterpart_method_name = (
233             clsname + '.' +
234             counterpart_method_unqualified)
235         native_method = ''
236
237     if pairing == ('C', 'J'):
238         counterpart_method_name = 'benchmark' + \
239             sequence_no
240
241     # 2. Set up calling sources.
242
243     if from_lang == 'C':
244         run_method = java_benchmark.native_run_method_t
245         native_method = ''
246
247     if from_lang == 'J':
248         run_method = put(
249             java_benchmark.java_run_method_t,
250             parameter_declarations="; ".join(
251                 parameter_declarations[0:i + 1]),
252             parameter_initialisations="; ".join(
253                 parameter_initialisations[0:i + 1]),
254             counterpart_method_name=(
255                 counterpart_method_name),
256             counterpart_method_arguments=(
257                 ", ".join(parameter_names[0:i + 1])))
258
259     if pairing == ('J', 'C'):
260         native_method = put(
261             java_benchmark.native_method_t,
262             modifiers=native_method_modifiers,
263
264             return_type=return_type[
265                 'java'],
266             name=native_method_name,
267             parameters=", ".join(
268                 parameter_declarations[0:i + 1]))
269
270     # 3. Common benchmark class wrapper for all
271     # benchmarks.
272
273     classname = benchmark_classname(
274         '2'.join(pairing), sequence_no)
275
276     nm = native_method if to_lang == 'C' else ''
277     java.append({
278         'filename': classname + ".java",
279         'class':
280             ''.join(
281                 packagename) + "." + classname,
282         'path': '/' + packagename,
283         'code': put(
284             java_benchmark.t,
285             representative=representative,
286             imports='',
287             has_dynamic_parameters='false',
288             is_allocating='false',
289             is_nonvirtual=(
290                 'false' if virtualcall else 'true'),

```

```

291         _id=benchmark_classname(
292             "", sequence_no),
293         description=spec[
294             'description'],
295         seq_no=class_counter,
296         from_language=from_lang,
297         to_language=to_lang,
298         class_relations='',
299         packagename='.'.join(
300             packagename),
301         classname=classname,
302         library_name=library_name,
303         run_method=run_method,
304         native_method=nm})
305
306     if (return_type.get('is-object') or
307         return_type.get('is-array')):
308         ret_expression = "{_type}Value".format(
309             _type=return_type['c'])
310     else:
311         ret_expression = parameter_initialisation(
312             'java', return_type, None)
313
314     # 4. Call target implementations.
315
316     if to_lang == 'J':
317         cmu = counterpart_method_unqualified
318         java_callees[cmu] = put(
319             java_counterparts.counterpart_t,
320             return_type=return_type[
321                 'java'],
322             privacy=target_modifier[0],
323             static=target_modifier[1],
324             methodname=cmu,
325             parameters=", ".join(
326                 parameter_declarations[0:i + 1]),
327             return_expression=ret_expression
328         )
329
330     if pairing == ('J', 'C'):
331         c_implementations.append(
332             put(c_nativemethod.t,
333                 return_type=return_type['c'],
334                 packagename=(
335                     '_').join(
336                         packagename),
337                 classname=classname,
338                 function=native_method_name,
339                 parameters=", ".join(
340                     c_parameter_declarations[0:i + 2]),
341                 return_expression=return_expression))
342
343     if pairing == ('C', 'C'):
344         jni_param_names = ['env', 'instance']
345         jni_param_names.extend(parameter_names)
346
347         c_runners.append(
348             put(c_nativemethod.t_caller_native,
349                 packagename='_'.join(packagename),
350                 classname=classname,

```

```

351         parameter_declarations=""; ".join(
352             c_parameter_declarations[1:i + 2]),
353         parameter_initialisations=""; ".join(
354             c_parameter_initialisations[
355                 0:i + 1]),
356         counterpart_method_name=(
357             "Java_{}_{_}{_}".format(
358                 ('_').join(packagename),
359                 ('J2CBenchmark' +
360                     str(sequence_no)),
361                 native_method_name),
362         counterpart_method_arguments="; ".join(
363             jni_param_names[0:i + 3]))
364
365     if pairing == ('C', 'J'):
366         if return_type.get('is-object',
367             return_type.get(
368                 'is-array', False)):
369             java_method_type = 'Object'
370         else:
371             java_method_type = return_type[
372                 'java'].capitalize()
373
374     arguments = ', '.join(parameter_names[0:i + 1])
375     if arguments != '':
376         arguments = ', ' + arguments
377
378     c_runners.append(
379         c_nativemethod.call_java_from_c(
380             static=(
381                 target_modifier[1] == 'static'),
382             nonvirtual=not virtualcall,
383             seq_no=class_counter,
384             packagename='_'.join(packagename),
385             classname=classname,
386             parameter_declarations="; ".join(
387                 c_parameter_declarations[1:i + 2]),
388             parameter_initialisations="; ".join(
389                 c_parameter_initialisations[
390                     0:i + 1]),
391             java_method_type=java_method_type,
392             call_variant='',
393             arguments=arguments))
394
395     c_methodid_inits.append(
396         put(c_module.mid_init_t,
397             seq_no=class_counter,
398             static=target_modifier[
399                 1].capitalize(),
400             method_name=counterpart_method_name,
401             method_descriptor=(
402                 jni_types.method_descriptor(
403                     return_type,
404                     type_combination[0:i + 1])))
405
406     ref_types = jni_types.object_types.values(
407 ) + jni_types.array_types.values()
408     jcp_decl = ''
409     jcp_init = ''
410     for _type in ref_types:

```

```

411         jcp_decl += "private static {_type1} {_type2}Value;\n".format(
412             _type1=_type['java'], _type2=_type['c'])
413         jcp_init += parameter_initialisation(
414             'java', _type, _type['c'] + 'Value') + ";\n"
415
416     java_counterparts_class = {
417         'filename': java_counterpart_classname + ".java",
418         'class': ''.join(
419             packagename) +
420         "." +
421         java_counterpart_classname,
422         'path': '/'.join(packagename),
423         'code': put(java_counterparts.t,
424                     packagename='.'.join(packagename),
425                     imports='',
426                     return_value_declarations=jcp_decl,
427                     return_value_inits=jcp_init,
428                     counterpart_methods=''.join(java_callees.values()))}
429
430     c_file = put(
431         c_module.t,
432         jni_function_templates=''.join(c_implementations),
433         initialisers='')
434
435     c_runners_file = put(
436         c_module.t,
437         jni_function_templates=''.join(c_runners),
438         initialisers=put(
439             c_module.initialisers_t,
440             mid_inits=''.join(c_methodid_inits),
441             amount_of_methods=class_counter))
442
443     return {'java': java,
444            'java_counterparts': java_counterparts_class,
445            'c': c_file,
446            'c_runners': c_runners_file}

```

jni_types.py

```

1  import itertools
2
3  primitive_types = None
4  object_types = None
5  other_types = None
6  types = None
7  return_types = None
8
9  representative_types = None
10
11 array_types = None
12
13 primitive_type_definitions = [
14     {
15         'symbol': 'b',
16         'java': 'boolean',

```

```

17     'c': 'jboolean',
18     'c-literal': '1',
19     'java-literal': 'true',
20     'jvm-desc': 'Z',
21     'byte_count': '1'
22 },
23
24 {
25     'symbol': 'y',
26     'java': 'byte',
27     'c': 'jbyte',
28     'c-literal': "'a'",
29     'java-literal': '(byte)100',
30     'jvm-desc': 'B',
31     'byte_count': '1',
32     'representative': True,
33 },
34
35 {
36     'symbol': 'c',
37     'java': 'char',
38     'c': 'jchar',
39     'c-literal': '12',
40     'java-literal': "'\u0012'",
41     'jvm-desc': 'C',
42     'byte_count': '2'
43 },
44
45 {
46     'symbol': 's',
47     'java': 'short',
48     'c': 'jshort',
49     'c-literal': '101',
50     'java-literal': '(short)101',
51     'jvm-desc': 'S',
52     'byte_count': '2'
53 },
54
55 {
56     'symbol': 'i',
57     'java': 'int',
58     'c': 'jint',
59     'c-literal': '102',
60     'java-literal': '102',
61     'jvm-desc': 'I',
62     'representative': True,
63     'byte_count': '4'
64 },
65
66 {
67     'symbol': 'l',
68     'java': 'long',
69     'c': 'jlong',
70     'c-literal': '103',
71     'java-literal': '103',
72     'jvm-desc': 'J',
73     'representative': True,
74     'byte_count': '8'
75 },
76

```



```

77     {
78         'symbol': 'f',
79         'java': 'float',
80         'c': 'jfloat',
81         'c-literal': '104.1',
82         'java-literal': '104.1f',
83         'jvm-desc': 'F',
84         'representative': True,
85         'byte_count': '4'
86     },
87
88     {
89         'symbol': 'd',
90         'java': 'double',
91         'c': 'jdouble',
92         'c-literal': '105.1',
93         'java-literal': '105.1',
94         'jvm-desc': 'D',
95         'representative': True,
96         'byte_count': '8'
97     },
98 ]
99
100 object_type_definitions = [
101
102     {
103         'symbol': 'O',
104         'java': 'Object',
105         'package': 'java.lang',
106         'c': 'jobject',
107         'c-literal': None,
108         'java-literal': None,
109         'is-object': True,
110         'representative': True,
111         'jvm-desc': 'Ljava/lang/Object;'
112     },
113
114     {
115         'symbol': 'C',
116         'java': 'Class',
117         'package': 'java.lang',
118         'c': 'jclass',
119         'c-literal': None,
120         'java-literal': None,
121         'is-object': True,
122         'jvm-desc': 'Ljava/lang/Class;'
123     },
124
125     {
126         'symbol': 'S',
127         'java': 'String',
128         'package': 'java.lang',
129         'c': 'jstring',
130         'c-literal': None,
131         'java-literal': '"a string"',
132         'is-object': True,
133         'jvm-desc': 'Ljava/lang/String;'
134     },
135
136     {

```

```

137         'symbol': 'T',
138         'java': 'Throwable',
139         'package': 'java.lang',
140         'c': 'jthrowable',
141         'c-literal': None,
142         'java-literal': None,
143         'is-object': True,
144         'jvm-desc': 'Ljava/lang/Throwable;'
145     }
146
147 ]
148
149 other_type_definitions = [
150
151     {
152         'symbol': 'v',
153         'java': 'void',
154         'c': 'void',
155         'c-literal': None,
156         'java-literal': None,
157         'representative': True,
158         'jvm-desc': 'V'
159     }
160 ]
161
162
163 def java_native_methodname(is_static, returntype, parametertypes):
164     ret = "_"
165     if is_static:
166         ret += "st_"
167     ret += types.get(returntype)['java'] + "_"
168     for t in parametertypes:
169         ret += types.get(t)
170
171
172 def java_native_methodsignature(is_static, returntype, parametertypes):
173     ret = "private"
174     if is_static:
175         ret += "static "
176
177 def type_combinations(size=0, typeset=None):
178     if size == 0:
179         size = len(typeset)
180
181     return list(itertools.islice(itertools.cycle(typeset), 0, size))
182
183
184 def method_descriptor(return_type, parameter_types):
185     return "({parameters}){returndesc}".format(
186         parameters=''.join([td['jvm-desc'] for td in parameter_types]),
187         returndesc=return_type['jvm-desc'])
188
189
190 def init_types():
191     global primitive_types, object_types, other_types, types
192     global return_types, array_types, representative_types
193
194     primitive_types = dict([(typedef['symbol'], typedef)
195                             for typedef in primitive_type_definitions])
196     object_types = dict([(typedef['symbol'], typedef)

```

```

197         for typedef in object_type_definitions])
198 other_types = dict([(typedef['symbol'], typedef)
199                     for typedef in other_type_definitions])
200
201 array_element_types = {}
202 array_element_types.update(primitive_types)
203 array_element_types['O'] = object_types['O']
204
205 array_types = dict(
206     [
207         ('A' + key,
208          {'symbol': 'A' + key,
209           'java': tipe['java'] + '[]',
210           'package': tipe.get('package', None),
211           'c': tipe['c'] + 'Array',
212           'c-literal': None,
213           'java-literal': None,
214           'is-array': True,
215           'representative': tipe.get('representative', False),
216           'java-element-type': tipe['java'],
217           'c-element-type': tipe['c'],
218           'jvm-desc': '[' + tipe['jvm-desc']
219          })
220     for key, tipe in array_element_types.iteritems()]
221
222 types = dict()
223 types.update(primitive_types)
224 types.update(object_types)
225 types.update(array_types)
226
227 return_types = dict()
228 return_types.update(types)
229 return_types.update(other_types)
230
231 # types.update(other_types)
232
233 init_types()

```

make_benchmarks.py

```

1 from benchmark_generator import generate_benchmarks, packagename
2 from make_custom_benchmarks import write_custom_benchmarks
3 from templates import java_registry_init
4 from templating import put
5
6 import sys
7 from sys import argv
8 import os.path
9 import os
10 import logging
11
12 # Log everything, and send it to stderr.
13 logging.basicConfig(level=logging.DEBUG)
14
15

```

```

16 def write_benchmark(benchmark, java_output_dir):
17     java_output_path = os.path.join(
18         java_output_dir,
19         benchmark["path"])
20
21     try:
22         os.makedirs(java_output_path)
23     except OSError:
24         pass
25
26     java_output = open(
27         os.path.join(
28             java_output_path,
29             benchmark["filename"]), 'w')
30
31     java_output.write(benchmark["code"])
32
33
34 def write_benchmarks(c_output, c_runners_output, java_output_dir):
35     benchmarks = generate_benchmarks()
36
37     c_output.write(benchmarks['c'])
38     c_runners_output.write(benchmarks['c_runners'])
39
40     write_benchmark(benchmarks['java_counterparts'], java_output_dir)
41     for benchmark in benchmarks['java']:
42         write_benchmark(benchmark, java_output_dir)
43
44     return [benchmark['class'] for benchmark in benchmarks['java']]
45
46
47 def write_benchmark_initialiser(classes):
48     benchmark_inits = []
49
50     for _class in classes:
51         benchmark_inits.append(java_registry_init.inits(_class))
52
53     path = os.path.join(
54         java_output_dir,
55         'fi/helsinki/cs/tituomin/nativebenchmark',
56         'BenchmarkInitialiser.java')
57
58     init_output = open(path, 'w')
59     init_output.write(
60         put(java_registry_init.t,
61            register_benchmarks="\n".join(benchmark_inits)))
62
63
64 if __name__ == "__main__":
65     try:
66         argv.pop(0)
67         c_output_name = argv.pop(0)
68         c_run_output_name = argv.pop(0)
69         c_custom_output_name = argv.pop(0)
70         java_output_dir = argv.pop(0)
71         c_definition_filename = argv.pop(0)
72         java_definition_filename = argv.pop(0)
73
74         definition_files = {
75             'C': open(c_definition_filename),

```

```

76         'J': open(java_definition_filename)}
77
78     c_run_output = open(c_run_output_name, 'w')
79     c_output = open(c_output_name, 'w')
80
81     classes = (write_benchmarks(c_output, c_run_output, java_output_dir) +
82               write_custom_benchmarks(
83                   definition_files,
84                   c_custom_output_name,
85                   java_output_dir))
86
87     write_benchmark_initialiser(classes)
88     print(", ".join(classes))
89     except Exception as e:
90         logging.exception("Exception was thrown.")
91         sys.exit(1)
92     else:
93         sys.exit(0)

```

make_custom_benchmarks.py

```

1  import re
2  import logging
3  from os import path
4  from sys import argv
5  import sys
6
7  from templating import put
8
9  from templates import arrays
10 from templates import loop_code
11 from templates import c_nativemethod
12 from templates import java_benchmark
13 from templates import java_registry_init
14
15 import jni_types
16
17 # Log everything, and send it to stderr.
18 logging.basicConfig(level=logging.DEBUG)
19
20 MAX_ALLOC_REPETITIONS = 500
21
22 i = re.IGNORECASE
23 begin_re = re.compile('s*//s*@begin\s*', flags=i)
24 end_re = re.compile('s*//s*@end\s*', flags=i)
25 inits_re = re.compile('s*//s*@inits-end\s*', flags=i)
26 benchmark_re = re.compile('s*//s*@(\s+)\s*')
27
28
29 def inits_block_end(line):
30     return inits_re.match(line)
31
32
33 def begins_block(line):
34     return begin_re.match(line)

```

```

35
36
37 def ends_block(line):
38     return end_re.match(line)
39
40
41 def is_benchmark_header(line):
42     return benchmark_re.match(line)
43
44
45 def parse_benchmark_header(line):
46     tokens = line.split()[1:]
47     b_properties = parse_properties(tokens[1:])
48     b_properties['id'] = tokens[0][1:]
49     return b_properties
50
51
52 def parse_properties(seq):
53     kvs = []
54     for s in seq:
55         splitted = s.split('=')
56         kvs.append((splitted[0], splitted[1]))
57     try:
58         return dict(kvs)
59     except ValueError as e:
60         print seq
61         print seq[0].split('=')
62         exit(1)
63
64
65 def abort_if_last(line):
66     if line == '':
67         logging.error("Invalid benchmark input file.")
68         exit(1)
69
70
71 def read_until(f, predicate, collect=None):
72     line = ''
73     while not predicate(line):
74         if collect is not None:
75             collect.append(line)
76             line = f.readline()
77             abort_if_last(line)
78     abort_if_last(line)
79     return line
80
81
82 def read_benchmarks(definition_files):
83     benchmarks = {}
84     for lang, f in definition_files.iteritems():
85         benchmarks[lang] = {'module': None, 'benchmarks': []}
86
87         module_start = []
88         inits = []
89         read_until(f, begins_block, collect=module_start)
90         read_until(f, inits_block_end, collect=inits)
91         benchmarks[lang]['inits'] = ''.join(inits)
92         benchmarks[lang]['module'] = ''.join(module_start)
93         line = read_until(f, is_benchmark_header)
94

```

```

95     while line != '':
96         bm_props = parse_benchmark_header(line)
97
98         bm_code = []
99         line = read_until(f,
100             lambda x: ends_block(x) or is_benchmark_header(x),
101             collect=bm_code)
102
103         bm_props['code'] = ''.join(bm_code)
104         benchmarks[lang]['benchmarks'].append(bm_props)
105
106         if ends_block(line):
107             break
108
109     add_field_and_array_benchmarks(benchmarks)
110     add_overhead_benchmarks(benchmarks)
111     return benchmarks
112
113 OVERHEAD_STEP = 2
114 OVERHEAD_STEPS = 11 # incl. zero
115 OVERHEAD_CODE_STATEMENT = "__a = (((__a * __a * __a) / __b) + __b) / __a; \n"
116
117
118 def add_overhead_benchmark(benchmarks, i, prefix, alloc):
119     overhead_code = []
120     for j in range(0, i):
121         overhead_code.append(OVERHEAD_CODE_STATEMENT)
122
123     benchmark = {
124         'code': ''.join(overhead_code),
125         'id': prefix + 'Overhead' + str(i).zfill(5),
126         'description': i
127     }
128
129     if alloc:
130         benchmark['alloc'] = True
131
132     c_b = benchmark.copy()
133     double_benchmark = benchmark.copy()
134     # double the amount of work for java (uses optimizations unlike c)
135     double_benchmark['code'] = ''.join(overhead_code + overhead_code)
136     j_b = double_benchmark
137     c_b['direction'] = 'cj'
138     j_b['direction'] = 'jj'
139     benchmarks['C']['benchmarks'].append(c_b)
140     benchmarks['J']['benchmarks'].append(j_b)
141
142
143 def add_overhead_benchmarks(benchmarks):
144     for i in range(1, OVERHEAD_STEPS * OVERHEAD_STEP, OVERHEAD_STEP):
145         for prefix, alloc in [('Alloc', True), ('Normal', False)]:
146             add_overhead_benchmark(benchmarks, i, prefix, alloc)
147     add_overhead_benchmark(benchmarks, 200, 'Warmup', False)
148
149
150 def macro_call(template, _type):
151     return template.format(
152         _type=_type['c'],
153         java_type_name=_type['java'].capitalize())
154

```

```

155
156 def make_id(template, _type):
157     return template.format(
158         _type=_type['java'].capitalize())
159
160
161 def add_field_and_array_benchmarks(benchmarks):
162     c = benchmarks['C']['benchmarks']
163     java = benchmarks['J']['benchmarks']
164
165     for _type in (
166         jni_types.primitive_types.values() +
167         [jni_types.object_types['O']]):
168         representative = _type.get('representative', False)
169
170         c.append({
171             'id': make_id('GetStatic{_type}Field', _type),
172             'representative': representative,
173             'direction': 'cj',
174             'code': macro_call(
175                 'GET_STATIC_TYPE_FIELD({_type}, {java_type_name});',
176                 _type)})
177
178         java.append({
179             'id': make_id('GetStatic{_type}Field', _type),
180             'representative': representative,
181             'direction': 'jj',
182             'class_init':
183                 'public {} persistentValue;'.format(_type['java']),
184             'method_init': '{} localPersistentValue = {};'.format(
185                 _type['java'], _type.get('java-literal') or 'objectValue'),
186             'code':
187                 ("localPersistentValue = "
188                  "mockObject.{_ctype}StaticField;"
189                  ).format(
190                     _javatype=_type['java'],
191                     _ctype=_type['c']
192                 ),
193             'finished': 'persistentValue = localPersistentValue;'
194         })
195
196         c.append({
197             'direction': 'cj',
198             'representative': representative,
199             'id': make_id('SetStatic{_type}Field', _type),
200             'code': macro_call(
201                 'SET_STATIC_TYPE_FIELD({_type}, {java_type_name});',
202                 _type)})
203
204         java.append({
205             'id': make_id('SetStatic{_type}Field', _type),
206             'representative': representative,
207             'direction': 'jj',
208             'code':
209                 "mockObject.{_ctype}StaticField = {_literal} ;".format(
210                     _ctype=_type['c'],
211                     _literal=_type.get('java-literal') or 'objectValue'
212                 ),
213         })
214

```



```

215
216 c.append({
217     'id': make_id('Get{_type}Field', _type),
218     'direction': 'cj',
219     'representative': representative,
220     'code': macro_call(
221         'GET_TYPE_FIELD({_type}, {java_type_name});',
222         _type)})
223
224 java.append({
225     'id': make_id('Get{_type}Field', _type),
226     'representative': representative,
227     'class_init':
228         'public {} persistentValue;'.format(_type['java']),
229     'method_init': '{ } localPersistentValue = {};'.format(
230         _type['java'], _type.get('java-literal') or 'objectValue'),
231     'direction': 'jj',
232     'code':
233         "localPersistentValue = mockObject.{_ctype}Field;".format(
234             _javatype=_type['java'],
235             _ctype=_type['c']
236
237         ),
238     'finished': 'persistentValue = localPersistentValue;'
239 })
240
241 c.append({
242     'id': make_id('Set{_type}Field', _type),
243     'direction': 'cj',
244     'representative': representative,
245     'code': macro_call(
246         'SET_TYPE_FIELD({_type}, {java_type_name});',
247         _type)})
248
249 java.append({
250     'id': make_id('Set{_type}Field', _type),
251     'representative': representative,
252     'direction': 'jj',
253     'code':
254         "mockObject.{_ctype}Field = {_literal} ;".format(
255             _ctype=_type['c'],
256             _literal=_type.get('java-literal') or 'objectValue'
257
258         ),
259 })
260
261 for _type in jni_types.primitive_types.values():
262     representative = _type.get('representative', False)
263     c.append({
264         'id': make_id('New{_type}Array', _type),
265         'representative': representative,
266         'direction': 'cj',
267         'vary': 'size',
268         'alloc': 'true',
269         'code': macro_call(
270             'NEW_PRIMITIVE_ARRAY({_type}, {java_type_name});',
271             _type)
272     })
273
274 # java

```

```

275
276 c.append({
277     'id': make_id('Get{_type}ArrayElements', _type),
278     'representative': representative,
279     'direction': 'cj',
280     'vary': 'size',
281     'code': macro_call(
282         ('GET_PRIMITIVE_ARRAY_ELEMENTS({_type}, {java_type_name});',
283          'RELEASE_PRIMITIVE_ARRAY_ELEMENTS'
284          '({_type}, {java_type_name});'),
285         _type)})
286
287 c.append({
288     'vary': 'size',
289     'direction': 'cj',
290     'representative': representative,
291     'id': make_id('Get{_type}ArrayRegion', _type),
292     'code': macro_call(
293         'GET_PRIMITIVE_ARRAY_REGION({_type}, {java_type_name});',
294         _type)})
295
296 c.append({
297     'vary': 'size',
298     'representative': representative,
299     'direction': 'cj',
300     'id': make_id('Set{_type}ArrayRegion', _type),
301     'code': macro_call(
302         'SET_PRIMITIVE_ARRAY_REGION({_type}, {java_type_name});',
303         _type)})
304
305 c.append({
306     'vary': 'size',
307     'representative': representative,
308     'direction': 'cj',
309     'id': make_id('Get{_type}ArrayLength', _type),
310     'code': macro_call(
311         'GET_PRIMITIVE_ARRAY_LENGTH({_type});',
312         _type)})
313
314 c.append({
315     'vary': 'size',
316     'representative': representative,
317     'direction': 'cc',
318     'id': make_id('ReadComplete{_type}Array', _type),
319     'code': put(
320         arrays.t_read,
321         declare_idx='jint idx;',
322         variable_in='%s__IN' % _type['c'],
323         array_variable='%s_buf__IN' % _type['c'],
324         element_literal=_type['c-literal']
325     )))
326
327 java_declarations = ('{0} {0}In;\n{0} [] {0}Arr = '
328                     'benchmarkParameter.retrieve{1}Array();').format(
329     _type['java'], _type['java'].capitalize())
330
331 java.append({
332     'vary': 'size',
333     'direction': 'jj',
334     'representative': representative,

```

```

335         'class_init': 'public int persistentValue;',
336         'method_init': 'int localPersistentValue = 0;',
337         'id': make_id('ReadComplete{_type}Array', _type),
338         'code': put(
339             arrays.t_read,
340             declare_idx='int idx;',
341             declare_variables=java_declarations,
342             variable_in='%sIn' % _type['java'],
343             array_variable='%sArr' % _type['java'],
344             element_literal=_type['java-literal']
345         ),
346         'finished': 'persistentValue = localPersistentValue;'
347     })
348
349     c.append({
350         'vary': 'size',
351         'direction': 'cc',
352         'representative': representative,
353         'id': make_id('WriteComplete{_type}Array', _type),
354         'code': put(
355             arrays.t_write,
356             declare_idx='jint idx;',
357             array_variable='%s_buf__IN' % _type['c'],
358             element_literal=_type['c-literal']
359         )})
360
361     java.append({
362         'vary': 'size',
363         'direction': 'jj',
364         'representative': representative,
365         'class_init': 'public int persistentValue;',
366         'method_init': 'int localPersistentValue = 0;',
367         'id': make_id('WriteComplete{_type}Array', _type),
368         'code': put(
369             arrays.t_write,
370             declare_variables=java_declarations,
371             declare_idx='int idx;',
372             # todo: writing affects other tests?
373             array_variable='%sArr' % _type['java'],
374             element_literal=_type['java-literal']),
375         'finished': 'persistentValue = localPersistentValue;'
376     })
377
378     # NIO variations of array/buffer reading/writing
379     if _type['java'] == 'boolean':
380         # Not available for booleans
381         continue
382
383     if _type['java'] == 'byte':
384         uppercase_typename = ''
385     else:
386         uppercase_typename = _type['java'].title()
387
388     # Read with hardcoded type method
389     java.append({
390         'vary': 'size',
391         'direction': 'jj',
392         'representative': True,
393         'class_init': 'public int persistentValue;',
394         'method_init': put(

```

```

395         arrays.t_init_nio,
396         type_declarations=java_declarations),
397     'id': make_id('ReadComplete{_type}NioByteBuffer', _type),
398     'code': put(
399         arrays.t_read_nio,
400         declare_idx='int idx;',
401         variable_in='%sIn' % _type['java'],
402         array_variable='directByteBufferValue',
403         type_name=uppercase_typename,
404         element_literal=_type['java-literal']),
405     'finished': 'persistentValue = localPersistentValue;'
406 })
407 # Write with hardcoded type method
408 java.append({
409     'vary': 'size',
410     'direction': 'jj',
411     'representative': True,
412     'class_init': 'public int persistentValue;',
413     'method_init': put(
414         arrays.t_init_nio,
415         type_declarations=java_declarations),
416     'id': make_id('WriteComplete{_type}NioByteBuffer', _type),
417     'code': put(
418         arrays.t_write_nio,
419         declare_variables='',
420         declare_idx='int idx;',
421         array_variable='directByteBufferValue',
422         type_name=uppercase_typename,
423         element_literal=_type['java-literal']),
424     'finished': 'persistentValue = localPersistentValue;'
425 })
426
427 declaration = java_declarations + "\n"
428 if _type['java'] == 'byte':
429     array_variable = 'directByteBufferValue'
430 else:
431     # Views are only relevant for non-byte types.
432     declaration += ('java.nio.{0}Buffer bufferView = '
433                    'directByteBufferValue.as{0}Buffer();').format(
434        uppercase_typename)
435     array_variable = 'bufferView'
436
437 # Bulk read through a typecast view buffer
438 java.append({
439     'vary': 'size',
440     'direction': 'jj',
441     'representative': True,
442     'class_init': 'public int persistentValue;',
443     'method_init': put(
444         arrays.t_init_nio,
445         type_declarations=declaration),
446     'id': make_id('ReadBulk{_type}NioByteBufferView', _type),
447     'code': put(
448         arrays.t_bulk_read,
449         array_variable=array_variable,
450         array_in='%sArr' % _type['java']),
451     'finished': 'persistentValue = localPersistentValue;'
452 })
453
454 # Bulk write through a typecast view buffer

```

```

455 java.append({
456     'vary': 'size',
457     'direction': 'jj',
458     'representative': True,
459     'class_init': 'public int persistentValue;',
460     'method_init': put(
461         arrays.t_init_nio,
462         type_declarations=declaration),
463     'id': make_id('WriteBulk[_type]NioByteBufferView', _type),
464     'code': put(
465         arrays.t_bulk_write,
466         array_variable=array_variable,
467         array_in='%sArr' % _type['java']),
468     'finished': 'persistentValue = localPersistentValue;'
469 })
470
471 if _type['java'] == 'byte':
472     continue
473
474 # Read through a typecast view buffer
475 java.append({
476     'vary': 'size',
477     'direction': 'jj',
478     'representative': True,
479     'class_init': 'public int persistentValue;',
480     'method_init': put(
481         arrays.t_init_nio,
482         type_declarations=declaration),
483     'id': make_id('ReadComplete[_type]NioByteBufferView', _type),
484     'code': put(
485         arrays.t_read_nio_as_view,
486         declare_idx='int idx;',
487         variable_in='%sIn' % _type['java'],
488         array_variable='bufferView',
489         type_name=uppercase_typename,
490         element_literal=_type['java-literal']),
491     'finished': 'persistentValue = localPersistentValue;'
492 })
493 # Write through a typecast view buffer
494 java.append({
495     'vary': 'size',
496     'direction': 'jj',
497     'representative': True,
498     'class_init': 'public int persistentValue;',
499     'method_init': put(
500         arrays.t_init_nio,
501         type_declarations=declaration),
502     'id': make_id('WriteComplete[_type]NioByteBufferView', _type),
503     'code': put(
504         arrays.t_write_nio_as_view,
505         declare_idx='int idx;',
506         array_variable='bufferView',
507         element_literal=_type['java-literal']),
508     'finished': 'persistentValue = localPersistentValue;'
509 })
510
511
512 def write_custom_benchmarks(
513     definition_files,
514     c_custom_output_name,

```

```

515     java_output_dir):
516 packagename = (
517     'fi',
518     'helsinki',
519     'cs',
520     'tituomin',
521     'nativebenchmark',
522     'benchmark')
523
524 all_benchmarks = read_benchmarks(definition_files)
525
526 out_c = open(c_custom_output_name, 'w')
527 out_c.write(all_benchmarks['C']['module'])
528
529 java_classes = {} # classname, contents
530
531 for lang, data in all_benchmarks.iteritems():
532     for benchmark in data['benchmarks']:
533
534         direction = benchmark['direction']
535         from_lang, to_lang = direction[0].upper(), direction[1].upper()
536         if from_lang != lang:
537             logging.error("Invalid language spec.")
538             exit(1)
539
540         classname = '{0}{2}{1}'.format(from_lang, to_lang) + benchmark['id']
541         if 'vary' in benchmark:
542             dyn_par = 'true'
543         else:
544             dyn_par = 'false'
545         if 'alloc' in benchmark:
546             # large heap 128/2 = 64 Mb, 128 el 8 byte array...
547             is_allocating = 'true'
548         else:
549             is_allocating = 'false'
550
551         representative = benchmark.get('representative', True)
552
553         if representative:
554             representative = "true"
555         else:
556             representative = "false"
557
558         if from_lang == 'C':
559             out_c.write(put(
560                 c_nativemethod.t_run_method,
561                 return_type='void',
562                 parameters='jobject instance',
563                 function='runInternal',
564                 packagename='_'.join(packagename),
565                 classname=classname,
566                 body=put(
567                     loop_code.t_c_jni_call,
568                     debug=classname,
569                     benchmark_body=benchmark['code'])))
570
571             java_classes[classname] = {
572                 'filename': classname + '.java',
573                 'code': (put(
574                     java_benchmark.t,

```

```

575         representative=representative,
576         _id=benchmark['id'],
577         packagename='.'.join(packagename),
578         classname=classname,
579         description=benchmark.get('description', ''),
580         is_allocating=is_allocating,
581         from_language=from_lang,
582         to_language=to_lang,
583         seq_no='-1',
584         has_dynamic_parameters=dyn_par,
585         is_nonvirtual='false',
586         run_method='public native void runInternal();',
587     ))}
588
589     elif from_lang == 'J' and to_lang == 'J':
590         java_classes[classname] = {
591             'filename': classname + '.java',
592             'code': (
593                 put(
594                     java_benchmark.t,
595                     representative=representative,
596                     _id=benchmark['id'],
597                     packagename='.'.join(packagename),
598                     imports="\n".join(
599                         ['import android.content.pm.PermissionInfo;',
600                          'import java.nio.ByteBuffer;',
601                          'import java.lang.ref.WeakReference;'],
602                     ),
603                     classname=classname,
604                     class_fields=benchmark.get('class_init', ''),
605                     description=benchmark.get('description', ''),
606                     is_allocating=is_allocating,
607                     from_language=from_lang,
608                     to_language=to_lang,
609                     is_nonvirtual='false',
610                     seq_no='-1',
611                     has_dynamic_parameters=dyn_par,
612                     run_method=put(
613                         java_benchmark.java_run_method_inline_t,
614                         init=data['inits'],
615                         type_init=benchmark.get('method_init', ''),
616                         loop=put(
617                             loop_code.t_java,
618                             finished=benchmark.get('finished', ''),
619                             benchmark_body=benchmark['code'])))
620
621 out_c.flush()
622 out_c.close()
623
624 for classname, contents in java_classes.iteritems():
625     f = open(
626         path.join(java_output_dir,
627             '%s'.join(packagename),
628             contents['filename']),
629         'w')
630     f.write(contents['code'])
631     f.flush()
632     f.close()
633
634 return ([ '%s'.join(packagename + (classname,))

```

```
for classname in java_classes.keys()]])
```

templates/arrays.py

```

1 from templating import partial
2
3 t_loop = """
4 <% declare_idx %>
5 <% declare_variables %>
6 for (idx = 0; idx < current_size; idx++) {
7     <% body %>
8 }
9 """
10
11 t_read = partial(
12     t_loop,
13     body="""
14     <% variable_in %> = <% array_variable %>[idx];
15     """)
16
17 t_write = partial(
18     t_loop,
19     body="""
20     <% array_variable %>[idx] = <% element_literal %>; """
21 )
22 t_init_nio = """
23     <% type_declarations %>
24     int localPersistentValue = 0;
25     current_size /= 64;
26
27 """
28
29 t_read_nio = partial(
30     t_loop,
31     body="""
32     <% variable_in %> = <% array_variable %>.get<% type_name %>(idx);
33     """)
34
35 t_write_nio = partial(
36     t_loop,
37     body="""
38     <% array_variable %>.put<% type_name %>(idx, <% element_literal %>);
39     """)
40
41 t_read_nio_as_view = partial(
42     t_loop,
43     body="""
44     <% variable_in %> = <% array_variable %>.get(idx);
45     """)
46
47 t_write_nio_as_view = partial(
48     t_loop,
49     body="""
50     <% array_variable %>.put(idx, <% element_literal %>);
51     """)

```



```

52
53 t_bulk_read = """
54 <% declare_variables %>
55 <% array_variable %>.clear();
56 <% array_variable %>.get(<% array_in %>, 0, current_size);
57 """
58
59 t_bulk_write = """
60 <% declare_variables %>
61 <% array_variable %>.clear();
62 <% array_variable %>.put(<% array_in %>, 0, current_size);
63 """

```

templates/c_jni_function.py

```

1 t = """
2
3 JNIEXPORT <% return_type %> JNICALL
4 Java_<% package %>_<% class_name %>_<% function %>
5 (JNIEnv *env, <% parameters %>) <%
6     <% set_returnvalues %>
7     %>
8
9 """

```

templates/c_module.py

```

1 t = """
2 #include <jni.h>
3 #include <android/log.h>
4 #include <stdio.h>
5 #include "natives.h"
6 #include "native_benchmarks.h"
7 #include "returnvalues.h"
8
9 <% initialisers %>
10 <% jni_function_templates %>
11
12 """
13
14 initialisers_t = """
15 static jmethodID mids[<% amount_of_methods %>];
16
17 static void init_methodids(JNIEnv *env) {
18     jmethodID mid;
19     <% mid_inits %>
20 }
21
22

```

```

23 int check_interrupted(JNIEnv *env) {
24     jobject current_thread = NULL;
25     current_thread = (
26         (*env)->CallStaticObjectMethod(env, thread_class, current_thread_mid));
27     if (current_thread == NULL) {
28         __android_log_write(ANDROID_LOG_ERROR, "check_interrupted",
29             "Can't get current thread");
30         return 1;
31     }
32     jboolean interrupted = (*env)->CallBooleanMethod(
33         env, current_thread, is_interrupted_mid);
34     (*env)->DeleteLocalRef(env, current_thread);
35     if (interrupted == JNI_TRUE) {
36         return 1;
37     }
38     return 0;
39 }
40
41 void throw_interrupted_exception(JNIEnv *env) {
42     jclass newExcCls;
43     newExcCls = (*env)->FindClass(env,
44         "java/lang/InterruptedException");
45     if (newExcCls == NULL) {
46         /* Unable to find the exception class, give up. */
47         return;
48     }
49     (*env)->ThrowNew(env, newExcCls, "thrown from C code");
50 }
51
52 JNIEXPORT void JNICALL
53 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_initNative
54 (JNIEnv *env, jclass cls, jlong reps, jlong interval, jclass javaCounterparts,
55 jobject javaCounterpartsInstance, jclass thread_cls) {
56     repetitions = reps;
57     interrupted = 0;
58
59     CHECK_INTERRUPTED_INTERVAL = interval;
60
61     jclass java_counterparts_class_global_ref = NULL;
62     jclass thread_class_global_ref = NULL;
63     jobject java_counterparts_object_global_ref = NULL;
64
65     java_counterparts_class_global_ref = (*env)->NewGlobalRef(
66         env, javaCounterparts);
67     if (java_counterparts_class_global_ref == NULL) {
68         __android_log_write(ANDROID_LOG_ERROR, "initNative",
69             "Could not create global ref.");
70         return;
71     }
72     java_counterparts_class = java_counterparts_class_global_ref;
73
74     java_counterparts_object_global_ref = (*env)->NewGlobalRef(env,
75         javaCounterpartsInstance);
76     if (java_counterparts_object_global_ref == NULL) {
77         __android_log_write(ANDROID_LOG_ERROR, "initNative",
78             "Could not create global ref.");
79         return;
80     }
81     java_counterparts_object = java_counterparts_object_global_ref;
82

```

```

83     if (!(*env)->IsInstanceOf(env, java_counterparts_object,
84         java_counterparts_class)) {
85         __android_log_write(ANDROID_LOG_ERROR, "initNative",
86             "JavaCounterparts instance or class is not correct.");
87         return;
88     }
89
90     init_methodids(env);
91
92     thread_class_global_ref = (*env)->NewGlobalRef(env, thread_cls);
93     if (thread_class_global_ref == NULL) {
94         __android_log_write(ANDROID_LOG_ERROR, "initNative",
95             "Could not create global ref.");
96         return;
97     }
98     thread_class = thread_class_global_ref;
99
100    current_thread_mid = (*env)->GetStaticMethodID(env, thread_class,
101        "currentThread", "()Ljava/lang/Thread;");
102    if (current_thread_mid == NULL) {
103        __android_log_write(ANDROID_LOG_ERROR, "initNative",
104            "Could not find currentThread");
105        return;
106    }
107    is_interrupted_mid = (*env)->GetMethodID(env, thread_class,
108        "isInterrupted", "()Z");
109    if (is_interrupted_mid == NULL) {
110        __android_log_write(ANDROID_LOG_ERROR, "check_interrupted",
111            "Can't get isInterrupted method");
112        return;
113    }
114
115 }
116
117 JNIEXPORT void JNICALL
118 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_setRepetitions
119 (JNIEnv *env, jclass cls, jlong reps) {
120     repetitions = reps;
121 }
122
123 JNIEXPORT void JNICALL
124 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_interruptNative
125 (JNIEnv *env, jclass cls) {
126     interrupted = 1;
127 }
128
129 JNIEXPORT void JNICALL
130 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_resetInterruptFlag
131 (JNIEnv *env, jclass cls) {
132     interrupted = 0;
133 }
134
135 """"
136
137 mid_init_t = """"
138     mid = (*env)->GetStaticMethodID(
139         env, java_counterparts_class, "<% method_name %>",
140         "<% method_descriptor %>");
141     if (mid == NULL) {
142         __android_log_write(ANDROID_LOG_VERBOSE, "nativemethod",

```

```

143         "<% method_descriptor %> not found.");
144     return; /* method not found */
145 }
146 mids[<% seq_no %> - 1] = mid;
147
148 """

```

templates/c_nativemethod.py

```

1 from templating import partial, put
2 import loop_code
3
4 t_run_method = """
5 JNIEXPORT <% return_type %> JNICALL
6 Java_<% packagename %>_<% classname %>_<% function %>
7 (JNIEnv *env, <% parameters %>) {
8     <% parameter_declarations %>;
9     <% parameter_initialisations %>;
10    <% prebody %>
11    <% body %>
12 }
13
14 """
15
16 t = partial(
17     t_run_method,
18     body='return <% return_expression %>;',
19     remove=['parameter_declarations',
20             'parameter_initialisations',
21             'prebody'])
22
23 # C to C
24 t_caller_native = partial(
25     t_run_method,
26     return_type='void',
27     function='runInternal',
28     parameters='jobject instance',
29     prebody='',
30     body=partial(
31         loop_code.t_c_jni_call,
32         benchmark_body=(
33             '<% counterpart_method_name %>' +
34             '(<% counterpart_method_arguments %>);'))
35
36 # C to J
37 t_caller_java = partial(
38     t_run_method,
39     return_type='void',
40     function='runInternal',
41     parameters='jobject instance',
42     prebody='jmethodID mid = mids[<% seq_no %> - 1];')
43
44
45 def call_java_from_c(static=True, nonvirtual=False, **parameters):
46     benchmark_body = ''

```

```

47     if static:
48         benchmark_body = (
49             '(*env)->CallStatic<% java_method_type %>Method<% call_variant %>',
50             '(env, java_counterparts_class, mid<% arguments %>);')
51     elif nonvirtual:
52         benchmark_body = (
53             '(*env)->CallNonvirtual',
54             '<% java_method_type %>Method<% call_variant %>',
55             '(env, java_counterparts_object, java_counterparts_class,',
56             ' mid<% arguments %>);')
57     else:
58         benchmark_body = (
59             '(*env)->Call<% java_method_type %>Method<% call_variant %>' +
60             '(env, java_counterparts_object, mid<% arguments %>);')
61
62     parameters['body'] = put(
63         loop_code.t_c_jni_call,
64         benchmark_body=put(benchmark_body, **parameters))
65
66     return partial(t_caller_java, **parameters)

```

templates/__init__.py

1

templates/java_benchmark.py

```

1 from templating import partial
2 import loop_code
3 import logging
4
5 t = """
6 package <% packagename %>;
7
8 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
9 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
10 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
11 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.BasicOption;
12 import fi.helsinki.cs.tituomin.nativebenchmark.MockObject;
13 <% imports %>
14 import android.util.Log;
15
16 public class <% classname %> <% class_relations %> extends Benchmark {
17
18     public <% classname %> (BenchmarkParameter bp) {
19         init(bp);
20     }
21
22     public String from() {

```

```

23         return "<% from_language %>";
24     }
25
26     public String to() {
27         return "<% to_language %>";
28     }
29
30     public int sequenceNo() {
31         return <% seq_no %>;
32     }
33
34     public String id() {
35         return "<% _id %>";
36     }
37
38     public boolean representative() {
39         return <% representative %>;
40     }
41
42     public boolean dynamicParameters() {
43         return <% has_dynamic_parameters %>;
44     }
45
46     public String description() {
47         return "<% description %>";
48     }
49
50     public boolean isAllocating() {
51         return <% is_allocating %>;
52     }
53
54     public boolean isNonvirtual() {
55         return <% is_nonvirtual %>;
56     }
57
58     <% class_fields %>
59
60     <% native_method %>
61
62     <% run_method %>
63
64 }
65
66 """
67
68 native_method_t = ('<% modifiers %> native <% return_type %> '
69                  '<% name %> (<% parameters %>);')
70 dynamic_parameter_t = (
71     'new BasicOption(BasicOption.VARIABLE, "<% variable %>").strip()
72     native_run_method_t = 'public native void runInternal();'
73
74 loop = partial(loop_code.t_java,
75                benchmark_body=('<% counterpart_method_name %> '
76                               '(<% counterpart_method_arguments %>);'))
77
78 java_run_method_t = partial("""
79
80     public void runInternal() {
81         JavaCounterparts counterpartInstance = JavaCounterparts.INSTANCE;
82         <% parameter_declarations %>;

```

```

83         <% parameter_initialisations %>;
84
85         <% loop %>
86     }
87
88     """, loop=loop)
89
90 java_run_method_inline_t = """
91
92     public void runInternal() {
93         <% init %>
94         <% type_init %>
95         <% loop %>
96     }
97
98     """

```

templates/java_counterparts.py

```

1  from templating import put
2
3  t = """
4  package <% packagename %>;
5
6  <% imports %>
7  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
8  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRunner;
9  import android.util.Log;
10
11
12  public enum JavaCounterparts {
13      INSTANCE;
14
15      <% return_value_declarations %>
16      public int persistentValue;
17      public static int staticpersistentValue = 0;
18
19      private JavaCounterparts() {
20          persistentValue = 0;
21      }
22
23      public static void initParams(BenchmarkParameter benchmarkParameter) {
24          <% return_value_inits %>
25      }
26
27      <% counterpart_methods %>
28  }
29  """
30
31  counterpart_t = """
32
33  <% privacy %> <% static %> <% return_type %> <% methodname %>(<% parameters %>) {
34      <% static %>persistentValue = (<% static %>persistentValue + 1) % 10;

```

```

37     return <% return_expression %>;
38 }
39
40 """
41
42 return_value_t = (
43     "private static <% actualtype %> = "
44     "benchmarkParameter.retrieve<% typename %>(<% typespecs %>);")

```

templates/java_registry_init.py

```

1  from templating import put
2
3  t = """
4  package fi.helsinki.cs.tituomin.nativebenchmark;
5
6  import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
7  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
8  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
9  import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.*;
10 import java.util.List;
11
12 public class BenchmarkInitialiser {
13
14     public static void init(BenchmarkParameter bp) {
15         List<Benchmark> benchmarks = BenchmarkRegistry.getBenchmarks();
16
17         <% register_benchmarks %>
18     }
19 }
20
21 """
22
23
24
25 def inits(classname):
26     return 'benchmarks.add(new {0} (bp));'.format(classname)

```

templates/loop_code.py

```

1  from templating import put, partial
2
3  t = """
4      <% declare_counters %>
5      <% additional_declaration %>
6
7      <% init_counters %>
8      division = repetitions / interval + 1;
9      remainder = repetitions % interval + 1;

```



```

10
11     <% debug %>
12     <% additional_init %>
13     while (--division != 0) {
14         <% init_counters %>
15         interval = interval + 1;
16         while (--interval != 0) {
17             <% pre_body %>
18             <% extra_debug %>
19             <% benchmark_body %>
20             <% post_body %>
21         }
22         if (<% test_interrupted %>) {
23             <% debug_interrupted %>
24             return;
25         }
26     }
27
28     <% additional_init %>
29
30     while (--remainder != 0) {
31         <% pre_body %>
32         <% benchmark_body %>
33         <% post_body %>
34     }
35
36     <% removal_prevention %>
37     <% finished %>
38
39     ""
40
41     jni_push_frame = ""
42     if (refs == 0) {
43         refs = LOCAL_FRAME_SIZE;
44         if ((*env)->PushLocalFrame(env, LOCAL_FRAME_SIZE) < 0) {
45             return;
46         }
47     }
48     ""
49
50     jni_pop_frame = ""
51     if (--refs == 0) {
52         (*env)->PopLocalFrame(env, NULL);
53     }
54
55     ""
56
57
58     t_c_base = partial(
59         t,
60         declare_counters='jlong interval, division, remainder;',
61         init_counters='interval = CHECK_INTERRUPTED_INTERVAL;',
62         test_interrupted='interrupted')
63
64     t_c_jni_call = partial(
65         t_c_base,
66         additional_declaration='jlong refs;',
67         additional_init='refs = 0;',
68         remove=[
69             'extra_debug',

```

```

70         'debug',
71         'debug_interrupted',
72         'removal_prevention'],
73     pre_body=jni_push_frame,
74     post_body=jni_pop_frame)
75
76 t_c = partial(
77     t_c_base,
78     remove=['extra_debug', 'debug', 'debug_interrupted',
79             'additional_declaration', 'additional_init',
80             'pre_body', 'post_body', 'removal_prevention'])
81
82 t_java = partial(
83     t,
84     test_interrupted='Thread.currentThread().isInterrupted()',
85     extra_debug='',
86     declare_counters='long interval, division, remainder;',
87     init_counters='interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL;',
88     removal_prevention='repetitionsLeft = division * interval + remainder;',
89     remove=['additional_declaration',
90             'additional_init',
91             'pre_body',
92             'post_body'])

```

templating.py

```

1  import string
2  import logging
3  formatter = string.Formatter()
4
5
6  class PartialDict(dict):
7
8      def __missing__(self, key):
9          return "<% " + key + " %>"
10
11
12  class PurgeDict(dict):
13
14      def __missing__(self, key):
15          return ""
16
17
18  def escape(string):
19      string = string.replace('{', '__BEG__')
20      string = string.replace('}', '__END__')
21      string = string.replace('<% ', '{')
22      string = string.replace(' %>', '}')
23      return string
24
25
26  def unescape(string):
27      string = string.replace('{', '<% ')
28      string = string.replace('}', ' %>')
29      string = string.replace('__BEG__', '{')

```

```

30     string = string.replace('__END__', '}')
31     return string
32
33
34 def put(template, remove=None, purge=True, **kwargs):
35     try:
36         template = escape(template)
37         for k, v in kwargs.iteritems():
38             if isinstance(v, str):
39                 kwargs[k] = escape(v)
40             if v is None:
41                 kwargs[k] = ''
42
43         if remove:
44             for k in remove:
45                 kwargs[k] = ''
46
47         if purge:
48             fdict = PurgeDict(**kwargs)
49         else:
50             fdict = PartialDict(**kwargs)
51
52         result = formatter.vformat(template, (), fdict)
53         result = unescape(result)
54         return result
55     except ValueError as e:
56         logging.error('error with template: ' + template)
57         raise e
58     return None
59
60
61 def partial(template, remove=None, **kwargs):
62     return put(template, remove=remove, purge=False, **kwargs)

```

Python-komponentit (analyysi)

Hakemistossa bench-analyzer.

/analysis.py

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 from numpy import polyfit, reshape, polyval
5
6 def linear_fit_columns(x, y):
7     p, residuals, rank, singular_values, rcond = polyfit(x, y, 1, full=True)
8     ynorm = normalized(x, y, p)
9     pnorm, residuals, rank, singular_values, rcond = polyfit(
10         x, ynorm, 1, full=True)
11     return p, residuals
12
13 def normalized(x, y, poly):
14     # normali
15     return (x - poly[1]) / poly[0]
16
17 def linear_fit(rows):
18     columns = reshape(rows, len(rows)*len(rows[0]), order='F').reshape(
19         (len(rows[0]), -1))
20     x = columns[0]
21     columns = columns[1:]
22     residuals = [linear_fit_columns(x, col)[1][0] for col in columns]
23     polys = [linear_fit_columns(x, col)[0] for col in columns]
24     return x, polys, residuals
25
26 def estimate_measuring_overhead(rows):
27     x, polys, residuals = linear_fit(rows)
28     return [p[1] for p in polys]
29
30 def optimize_bins(x):
31     """
32     Created on Thu Oct 25 11:32:47 2012
33
34     Histogram Binwidth Optimization Method
35
36     Shimazaki and Shinomoto, Neural Comput 19 1503-1527, 2007
37     2006 Author Hideaki Shimazaki, Matlab
38     Department of Physics, Kyoto University
39     shimazaki at ton.scphys.kyoto-u.ac.jp
40     Please feel free to use/modify this program.
41
42     Version in python adapted Érbet Almeida Costa
43
44     Data: the duration for eruptions of
45     the Old Faithful geyser in Yellowstone National Park (in minutes)
46     or normal distribution.
47     Version in python adapted Érbet Almeida Costa
48     Bugfix by Takuma Torii 2.24.2013
49
50     """
```

```

51
52 import numpy as np
53 from numpy import mean, size, zeros, where, transpose
54 from numpy.random import normal
55 from matplotlib.pyplot import hist
56 from scipy import linspace
57 import array
58
59 x_max = max(x)
60 x_min = min(x)
61 N_MIN = 4      #Minimum number of bins (integer)
62                #N_MIN must be more than 1 (N_MIN > 1).
63 N_MAX = 1000   #Maximum number of bins (integer)
64 N = range(N_MIN,N_MAX) # #of Bins
65 N = np.array(N)
66 D = (x_max-x_min)/N    #Bin size vector
67 C = zeros(shape=(size(D),1))
68
69 #Computation of the cost function
70 for i in xrange(size(N)):
71     edges = linspace(x_min,x_max,N[i]+1) # Bin edges
72     ki = hist(x,edges) # Count # of events in bins
73     ki = ki[0]
74     k = mean(ki) #Mean of event count
75     v = sum((ki-k)**2)/N[i] #Variance of event count
76     C[i] = (2*k-v)/((D[i])**2) #The cost Function
77 #Optimal Bin Size Selection
78
79 cmin = min(C)
80 idx = where(C==cmin)
81 idx = int(idx[0])
82 optD = D[idx]
83
84 edges = linspace(x_min,x_max,N[idx]+1)
85
86 return optD, edges

```

/datafiles.py

```

1 #!/usr/bin/python
2
3 import re
4 from collections import OrderedDict as odict
5 import sys
6
7 SEPARATOR = ','
8 RE_EMPTY = re.compile('^\\s*$')
9 RE_NUMERICAL = re.compile('^-[0-9]+$')
10
11
12 def explode(line):
13     return line.split(SEPARATOR)
14
15 def value(string, key=None):
16     if key in ['start', 'end']:

```

```

17         return string
18     if key == 'class':
19         return string.split('.')[ -1]
20     if string == '-' or RE_EMPTY.match(string):
21         return None
22     if RE_NUMERICAL.match(string):
23         return int(string)
24     else:
25         return string
26
27 def empty_label():
28     empty_label.cnt += 1
29     return 'empty_{0}'.format(empty_label.cnt)
30
31 empty_label.cnt = 0
32
33 def read_datafiles(files, silent=False):
34     if not silent:
35         print 'Reading from %s files' % len(files)
36     benchmarks = []
37     #-1: there is an empty field at the end...
38
39     keys_with_values = set()
40     all_keys = set()
41
42     lineno = 1
43     for i, f in enumerate(files):
44         line = f.readline()
45         labels = explode(line)
46         for i, l in enumerate(labels):
47             # account for the fact that there might be an empty label
48             # and corresponding column (usually the last)
49             if RE_EMPTY.match(l):
50                 labels[i] = empty_label()
51
52         all_keys.update(labels)
53
54         line = f.readline()
55         while line != '':
56             exploded_line = explode(line)
57             pad_amount = len(labels) - len(exploded_line)
58             exploded_line.extend(['-'] * pad_amount)
59             if len(labels) != len(exploded_line):
60                 print ('missing values', f.name, 'line', lineno, 'labels',
61                       len(labels), 'values', len(exploded_line))
62                 exit(1)
63
64             benchmark = dict()
65             benchmark['lineno'] = lineno
66
67             for key, string in zip(labels, exploded_line):
68                 benchmark[key] = value(string, key=key)
69
70                 if value(string, key=key) != None:
71                     keys_with_values.add(key)
72
73             #if benchmark['response_time'] != None:
74                 benchmarks.append(benchmark)
75
76             line = f.readline()

```

```

77         lineno += 1
78
79     keys_without_values = all_keys - keys_with_values
80
81     benchmark_keycount = None
82     for benchmark in benchmarks:
83         for key in keys_without_values:
84             if key in benchmark:
85                 del benchmark[key]
86             current_keycount = len(benchmark.keys())
87             benchmark_keycount = benchmark_keycount or current_keycount
88             if benchmark_keycount != current_keycount:
89                 print ("Benchmarks have different amount of data",
90                        benchmark_keycount, current_keycount, "at line",
91                        benchmark['lineno'])
92             exit(1)
93
94     if not silent:
95         print 'Read %d lines' % (lineno - 1)
96     return benchmarks
97
98 def read_measurement_metadata(mfile, combine_compatibles):
99     compatibles = odict()
100     measurement = None
101     line = None
102
103     i = 0
104     while line != '':
105         skipped = False
106         while line == "\n":
107             line = mfile.readline()
108             skipped = True
109
110         if skipped:
111             if measurement:
112                 if 'tools' in measurement:
113                     measurement['tool'] = measurement['tools']
114                     revision = measurement.get('code-revision')
115                     checksum = measurement.get('code-checksum')
116                     repetitions = measurement.get('repetitions')
117                     tool = measurement.get('tool')
118                     cpufreq = measurement.get('cpu-freq')
119                     benchmark_set = measurement.get('benchmark-set')
120                     substring_filter = measurement.get('substring-filter')
121                     if measurement.get('rounds') == None:
122                         measurement['rounds'] = 1
123
124                 if revision and repetitions:
125                     if combine_compatibles:
126                         key = (revision, checksum, repetitions, tool, cpufreq,
127                               benchmark_set, substring_filter)
128                     else:
129                         key = i
130                         i += 1
131                     if key not in compatibles:
132                         compatibles[key] = []
133                     compatibles[key].append(measurement)
134                 measurement = {}
135
136         if line != None:

```

```

137         splitted = line.split()
138         if len(splitted) > 1:
139             key = splitted[0].rstrip(':')
140             val = ' '.join(splitted[1:])
141             measurement[key] = val.strip()
142
143     line = mfile.readline()
144
145     return compatibles

```

/gnuplot.py

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import os
5 import uuid
6
7 INIT_PALETTE = """
8 # line styles for ColorBrewer Dark2
9 # for use with qualitative/categorical data
10 # provides 8 dark colors based on Set2
11 # compatible with gnuplot >=4.2
12 # author: Anna Schneider
13
14 # line styles
15 set style line 1 pt 7 lt 1 lc rgb '#1B9E77' # dark teal
16 set style line 2 pt 7 lt 1 lc rgb '#D95F02' # dark orange
17 set style line 3 pt 7 lt 1 lc rgb '#7570B3' # dark lilac
18 set style line 4 pt 7 lt 1 lc rgb '#000000' # black
19 set style line 5 pt 7 lt 1 lc rgb '#E7298A' # dark magenta
20 set style line 6 pt 7 lt 1 lc rgb '#66A61E' # dark lime green
21 set style line 7 pt 7 lt 1 lc rgb '#E6AB02' # dark banana
22 set style line 8 pt 7 lt 1 lc rgb '#A6761D' # dark tan
23 set style line 9 pt 7 lt 1 lc rgb '#666666' # dark gray
24 set style line 10 pt 7 lt 1 lc rgb '#1b70b3' # dark blue
25
26 set style line 11 pt 5 lt 2 lc rgb '#1B9E77' # dark teal
27 set style line 12 pt 5 lt 2 lc rgb '#D95F02' # dark orange
28 set style line 13 pt 5 lt 2 lc rgb '#7570B3' # dark lilac
29 set style line 14 pt 5 lt 2 lc rgb '#000000' # black
30 set style line 15 pt 5 lt 2 lc rgb '#E7298A' # dark magenta
31 set style line 16 pt 5 lt 2 lc rgb '#66A61E' # dark lime green
32 set style line 17 pt 5 lt 2 lc rgb '#E6AB02' # dark banana
33 set style line 18 pt 5 lt 2 lc rgb '#A6761D' # dark tan
34 set style line 19 pt 5 lt 2 lc rgb '#666666' # dark gray
35 set style line 20 pt 5 lt 2 lc rgb '#1b70b3' # dark blue
36
37
38 # palette
39 set palette maxcolors 8
40 set palette defined ( 0 '#1B9E77',\
41                     1 '#D95F02',\
42                     2 '#7570B3',\
43                     3 '#E7298A',\

```



```

44         4 '#66A61E',\
45         5 '#E6AB02',\
46         6 '#A6761D',\
47         7 '#666666' )
48 ""
49
50 INIT_PLOTS_PDF = ""
51 set terminal pdfcairo size 32cm,18cm {sizesuffix}
52 set size 1, 0.95
53 set output '{filename}'
54 ""
55
56 INIT_PLOTS_LATEX = ""
57 set terminal epslatex input color \
58 header "\\\caption{{{caption}}}\label{{fig:{label}}}" {sizesuffix}
59 set pointsize 1.0
60 set format y "%4.2s%cs"
61 set output
62 ""
63
64 INIT_PLOTS_SVG = ""
65 set terminal svg {sizesuffix}
66 set pointsize 1.0
67 set format y "%4.2s%cs"
68 set output
69 ""
70
71 INIT_PLOTS_COMMON = ""
72 set grid
73 set xlabel "kutsuparametrien määrä"
74 ""
75
76 INIT_PLOT_LABEL_PDF = ""
77 set label 1 "{bid}" at graph 0.01, graph 1.06
78 ""
79
80 TEMPLATES = {}
81 INIT_KEY = {}
82
83 TEMPLATES['binned_init'] = ""
84 set title '{title}'
85 binwidth={binwidth}
86 set boxwidth binwidth
87 set style fill solid 1.0
88 set xrange [{min_x}:{max_x}]
89 set yrange [0:{max_y}]
90 ""
91 # border lt -1
92 #bin(x,width)=width*floor(x/width) + width/2.0
93
94 TEMPLATES['binned_frame'] = ""
95 #set label 2 "{datapoints}" at graph 0.8, graph 1.06
96 set bmargin 20
97 set tmargin 20
98 set rmargin 20
99 set lmargin 20
100 plot '-' using 1:2 notitle with boxes lt rgb "{color}"\n{values}\ne\n
101 #unset xlabel
102 #unset ylabel
103 #unset label 1

```

```

104 #unset title
105 unset xtics
106 unset ytics
107 ""
108
109 SET_TITLE_AND_PAGE_LABEL = ""
110 set title '{title}'
111 set label 2 "{page}" at screen 0.9, screen 0.95
112 ""
113
114 INIT_KEY['simple_groups'] = ""
115 set key {key_placement} box notitle width -3 height +1 vertical
116 ""
117
118 TEMPLATES['simple_groups'] = ""
119 set ylabel "vasteaika {reps} toistolla"
120 set xlabel "{xlabel}"
121 plot for [I=2:{last_column}] '{filename}' index {index} \
122 using 1:I title columnhead with points ls I-1
123 ""
124
125 TEMPLATES['fitted_lines'] = ""
126 set ylabel "vasteaika {reps} toistolla"
127 set xlabel "{xlabel}"
128 plot for [I=2:{last_real_column}] '{filename}' index {index} using 1:I \
129 title columnhead with points ls I-1, \
130 for [I={first_fitted_column}:{last_column}] '{filename}' index {index} \
131 using 1:I notitle with lines ls I-{first_fitted_column}+1
132 ""
133
134 TEMPLATES['named_columns'] = ""
135 set yrange [0:∗]
136 set xlabel "{xlabel}"
137 plot for [I=2:{last_column}] '{filename}' index {index} using I:xtic(1) \
138 title columnhead with linespoints
139 ""
140
141 TEMPLATES['histogram'] = ""
142 unset xlabel
143 unset ylabel
144 set y2label "vasteaika {reps} toistolla"
145 set size 1, 1
146 unset x2tics
147 #unset xtics
148 unset ytics
149
150 set y2tics format "%.00s%cs" rotate
151
152 set xtics out rotate
153 set key at graph 0.1, 0.9 width 2 height 8 notitle horizontal nobox samplen 0.2
154 set label 1 'C$\\rightarrow$Java' at graph 0.145, 0.78 left rotate by 90
155 set label 2 'Java$\\rightarrow$Java' at graph 0.205, 0.78 left rotate by 90
156 set style data histograms
157 set style histogram clustered
158
159 plot [] [0:∗] for [I=2:{last_column}] '{filename}' index {index} \
160 using I:xtic(1) every ::1 title " " with histogram fillstyle solid 1.0 border lt -1
161 ""
162
163 measurement_id = None

```

```

164 plot_directory = '/home/tituomin/gradu/paper/figures/plots'
165
166 def init(plotscript, filename, mid, output_type='pdf'):
167     global measurement_id, plot_directory
168     measurement_id = mid
169     if output_type == 'pdf':
170         plotscript.write(INIT_PLOTS_PDF.format(filename=filename))
171         plotscript.write(INIT_PLOT_LABEL_PDF.format(bid=measurement_id))
172     plotscript.write(INIT_PLOTS_COMMON)
173     plotscript.write(INIT_PALETTE)
174
175 GROUPTITLES={
176     'direction': 'kutsusuunta',
177     'from': 'kieli'
178 }
179
180 def output_plot(data_headers, data_rows, plotpath,
181                 plotscript, title, specs, style, page,
182                 identifier,
183                 xlabel, additional_data=None, output='pdf',
184                 key_placement="inside top left", reps='XXX-fixme-XXX'):
185     global plot_directory
186     template = TEMPLATES[style]
187
188     rowlen = len(data_rows[0]) - 1
189     size = 'normal'
190     if style == 'fitted_lines':
191         rowlen /= 2
192     if (page > 51 and rowlen > 7) or rowlen > 10:
193         size = 'tall'
194     if rowlen < 15:
195         size = 'normal'
196     if identifier in [
197         'basic-call-all-types-j-j-fit',
198         'basic-call-all-types-c-c-fit',
199         'variable-argument-size-j-c',
200         'special-calls-arrayelements-c-j-fit',
201         'special-calls-arrayregion-c-j-fit']:
202         size = 'tall'
203
204     if output in ['latex', 'svg']:
205         if output == 'latex':
206             init_tmpl = INIT_PLOTS_LATEX
207             file_suffix = 'tex'
208         elif output == 'svg':
209             init_tmpl = INIT_PLOTS_SVG
210             file_suffix = 'svg'
211         sizesuffix=''
212         if size == 'tall':
213             if output == 'svg':
214                 sizesuffix = 'size 1000,800'
215             else:
216                 sizesuffix="size 15cm,13cm"
217         else:
218             if output == 'svg':
219                 sizesuffix="size 1000,600"
220             else:
221                 sizesuffix="size 15cm,10cm"
222     plotscript.write(
223         init_tmpl.format(

```

```

224         caption=title,
225         label=identifier,
226         sizesuffix=sizesuffix))
227     if specs['variable'] == 'dynamic_size':
228         plotscript.write("set xrange [0:512]\n")
229         plotscript.write("set xtics 0, 64\n")
230         plotscript.write("set format x \"%6.sB\"\n")
231     else:
232         plotscript.write("unset xtics\n")
233         plotscript.write("set xtics autofreq\n")
234         plotscript.write("set xrange [*:*]\n")
235         plotscript.write("set format x \"%6.s\"\n")
236
237     if size == 'tall':
238         if identifier in ['special-calls-arrayelements-c-j-fit',
239                           'special-calls-arrayregion-c-j-fit']:
240             plotscript.write(
241                 "set tmargin at screen 0.8\nset key above box "
242                 "horizontal maxrows 8 maxcols 4 samplen 1 "
243                 "spacing .5 font \",4\"\n");
244         else:
245             plotscript.write(
246                 "set tmargin at screen 0.85\n"
247                 "set key above nobox horizontal\n");
248     else:
249         plotscript.write("set tmargin at screen 0.95\n")
250     plotscript.write("set output '{}'.format(
251         os.path.join(plot_directory,
252                       "plot-{}-{}.{}".format(
253                         measurement_id, identifier, file_suffix))))
254
255     if plotpath:
256         # external data
257         filename = os.path.join(plotpath, "plot-" + str(uuid.uuid4()) + ".data")
258         plotdata = open(filename, 'w')
259         specs['convert_to_seconds'] = False # (output == 'latex')
260         if output == 'latex':
261             specs['tinylabels'] = True
262         if output == 'svg':
263             specs['scriptlabels'] = True
264         plotdata.write(print_benchmarks(data_headers, data_rows, title,
265                                         **specs))
266
267     miny = 0
268     for row in data_rows:
269         for cell in row[1:]:
270             if cell < miny:
271                 miny = cell
272     if miny == None:
273         miny = '*'
274
275     if output == 'pdf':
276         plotscript.write(SET_TITLE_AND_PAGE_LABEL.format(page=identifier,
277                                                         title=title))
278
279     if style == 'binned':
280         plotscript.write(template.format(
281             title = title, page = identifier, filename = filename, index = 0,
282             last_column = len(data_rows[0]),
283             xlabel = xlabel, miny=miny, **additional_data))

```

```

284
285 elif style == 'fitted_lines':
286     length = len(data_headers) - 1
287     last_real_column = 1 + length / 2
288     first_fitted_column = last_real_column + 1
289     plotscript.write(template.format(
290         title = title, reps = reps, page = identifier, filename = filename,
291         index = 0, last_column = len(data_rows[0]),
292         xlabel = xlabel, miny=miny, last_real_column=last_real_column,
293         first_fitted_column=first_fitted_column))
294
295 elif style == 'simple_groups':
296     grouptitle = GROUPTITLES.get(specs['group'], 'group')
297     if key_placement is None:
298         plotscript.write("\nunset key\n")
299     elif size != 'tall':
300         plotscript.write(INIT_KEY[style].format(
301             key_placement=key_placement))
302
303     plotscript.write(template.format(
304         title = title, reps = reps, page = identifier, filename = filename,
305         index = 0, last_column = len(data_rows[0]),
306         xlabel = xlabel, miny=miny, grouptitle=grouptitle))
307
308 else:
309     grouptitle = GROUPTITLES.get(specs['group'], 'group')
310     plotscript.write(template.format(
311         title = title, page = identifier, filename = filename, index = 0,
312         last_column = len(data_rows[0]),
313         key_placement = key_placement, xlabel = xlabel, reps=reps,
314         miny=miny, grouptitle=grouptitle))
315
316
317 def print_benchmarks(data_headers, data_rows, title, group=None, variable=None,
318                     measure=None, convert_to_seconds=False, tinylabels=False,
319                     scriptlabels=False):
320     result = '#{0}\n'.format(title)
321     if group and variable and measure:
322         result = '#measure:{m} variable:{v} group:{g}'.format(
323             m=measure, v=variable, g=group)
324
325     prefix = ""
326     suffix = ""
327     if tinylabels:
328         prefix = "\\|\\|\\|tiny "
329     elif scriptlabels:
330         prefix = "\\|\\|\\|tiny{"
331         suffix = "}"
332     result = " ".join([format_value("{}{}{}".format(prefix, k, suffix))
333                        for k in data_headers])
334     result += '\n'
335
336     for row in data_rows:
337         results = []
338         for i, v in enumerate(row):
339             convert = convert_to_seconds and i > 0
340             results.append(format_value(v, convert_to_seconds=convert))
341         result += ' '.join(results) + '\n'
342     result += '\n\n'
343

```

```

344     return result
345
346 def format_value(value, convert_to_seconds=False):
347     if value == None:
348         return "-500"
349     if type(value) == str:
350         return "{0}".format(value)
351     if type(value) == int:
352         strval = str(value)
353         if convert_to_seconds == False:
354             return strval
355         strval = strval.zfill(10)
356         strlen = len(strval)
357         return "{0}.{1}".format(
358             strval[0:strlen-9],
359             strval[strlen-9:])
360
361     return str(value)
362
363 def hex_color_gradient(start, end, point):
364     # start, end are tuples with r,g,b values (integer)
365     # point is a point between 0 (start) and 1000 (end)
366     return "#" + "".join(
367         "{:0>2X}".format(
368             int(start[i] +
369                 ((end[i] - start[i]) * (float(point))))))
370         for i in range(0,3))

```

/plot_data.py

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 from collections import OrderedDict as odict
5 from itertools import groupby
6 from subprocess import call
7 from sys import argv
8 import functools
9 import pprint
10 import re
11 import os
12 import sys
13 import shutil
14 import uuid
15
16 import glob
17 import zipfile
18
19 import numpy
20 from numpy import array
21
22 from jni_types import primitive_type_definitions
23 from jni_types import object_type_definitions, array_types
24 from datafiles import read_datafiles, read_measurement_metadata
25 import analysis

```

```

26 from analysis import linear_fit, estimate_measuring_overhead
27 import gnuplot
28 import textualtable
29
30 FNULL = None
31
32 primitive_types = [
33     t['java']
34     for t in primitive_type_definitions
35 ]
36
37 reference_types = [
38     t['java']
39     for t in array_types.itervalues()
40 ]
41
42 reference_types.extend([
43     t['java']
44     for t in object_type_definitions
45 ])
46
47 types = reference_types + primitive_types
48
49 plot_axes = {
50     'description': 'operaatioiden määrä',
51     'parameter_count': 'kutsuparametrien määrä',
52     'dynamic_size': 'kohteen koko',
53     'direction': 'kutsusuunta',
54     'id': 'nimi'
55 }
56 pp = pprint.PrettyPrinter(depth=10, indent=4)
57
58 debugdata = open('/tmp/debug.txt', 'w')
59
60 def format_direction(fr, to, latex):
61     if fr == 'J':
62         fr = 'Java'
63     if to == 'J':
64         to = 'Java'
65     if latex:
66         SEPARATOR = '$\\\\\\rightarrow$'
67     else:
68         SEPARATOR = ' > '
69     return "%s%s%s" % (fr, SEPARATOR, to)
70
71 DIRECTIONS = [('C', 'J'), ('J', 'C'), ('J', 'J'), ('C', 'C')]
72
73 def preprocess_benchmarks(benchmarks, global_values, latex=None):
74     # For allocating benchmarks, the repetition count for individual benchmarks
75     # come from the datafile. For non-allocating, it is a global value.
76     keys = set([key for b in benchmarks for key in b.keys()])
77     if 'repetitions' in keys:
78         benchmarks = [b for b in benchmarks if b['repetitions'] is not None]
79     for b in benchmarks:
80         add_derived_values(b, latex=latex)
81         add_global_values(b, global_values)
82     return benchmarks
83
84 def add_derived_values(benchmark, latex=None):
85     if benchmark.get('response_time_millis') != None:

```

```

86     benchmark['response_time'] = benchmark.get('response_time_millis')
87     benchmark['time_unit'] = 'milliseconds'
88     del benchmark['response_time_millis']
89     if benchmark.get('dynamic_size') == None:
90         benchmark['dynamic_variation'] = 0
91         benchmark['dynamic_size'] = 0
92     else:
93         benchmark['dynamic_variation'] = 1
94     if benchmark['no'] == -1:
95         # Custom benchmark, do some name mapping:
96         bid = benchmark['id']
97         rename = True
98         if bid == 'CopyUnicode':
99             bid = 'GetStringRegion'
100         elif bid == 'CopyUTF':
101             bid = 'GetStringRegionUTF'
102         elif bid == 'StringLength':
103             bid = 'GetStringLength'
104         elif bid == 'StringLengthUTF':
105             bid = 'GetStringUTFLength'
106         elif bid == 'ReadUnicode':
107             bid = 'ReadString'
108         elif bid == 'ReadUnicodeCritical':
109             bid = 'ReadStringCritical'
110         elif bid == 'ReadUTF':
111             bid = 'ReadStringUTF'
112         elif bid == 'ReadUtf':
113             bid = 'ReadStringUTF'
114         elif bid == 'ReadObjectArrayElement':
115             bid = 'GetObjectArrayElement'
116         elif bid == 'WriteObjectArrayElement':
117             bid = 'SetObjectArrayElement'
118         else:
119             rename = False
120         if rename:
121             benchmark['id'] = bid
122
123     single_type = None
124     if (benchmark.get('parameter_count') == 0):
125         single_type = 'any'
126     elif (benchmark.get('parameter_type_count') == 1):
127         for tp in types:
128             if benchmark.get('parameter_type_{t}_count'.format(t=tp)) != None:
129                 single_type = tp
130                 break
131     benchmark['direction'] = format_direction(
132         benchmark['from'], benchmark['to'], latex)
133     benchmark['single_type'] = single_type
134     if 'Nio' in benchmark['id']:
135         benchmark['nio'] = True
136     else:
137         benchmark['nio'] = False
138
139 def add_global_values(benchmark, global_values):
140     for key, val in global_values.iteritems():
141         if key not in benchmark or benchmark[key] == None:
142             benchmark[key] = val
143         elif key == 'multiplier' and benchmark[key] != None:
144             benchmark[key] *= val
145

```



```

146
147 def extract_data(benchmarks,
148                  group=None, variable=None, measure=None,
149                  min_series_length=2, sort=None, min_series_width=None):
150
151     # info == extra metadata not to be analyzed
152     info = ['no', 'from', 'to', 'lineno', 'start', 'end']
153
154     if 'class' in benchmarks[0]:
155         info.append('class')
156     if 'description' in benchmarks[0]:
157         info.append('description')
158     if re.match('parameter_type_.+count', variable):
159         info.append('parameter_count')
160     if variable != 'id':
161         info.append('id')
162
163     # note: all the benchmarks have the same keyset
164     all_keys = set(benchmarks[0].keys())
165
166     # the actual keys of interest must have the least weight in sorting
167     sort_last = [group, variable, measure] + info
168     controlled_variables = all_keys - set(sort_last)
169     sorted_keys = list(controlled_variables) + sort_last
170
171     sorted_benchmarks = sorted(
172         benchmarks,
173         cmp=functools.partial(comp_function, sorted_keys))
174
175     # 1. group benchmarks into a multi-dimensional list
176     # with the following structure:
177     # - compatible-measurements (controlled variables are equal)
178     # - plots (list of individual data series ie. plots)
179     # - multiple measurements ()
180     benchmarks = group_by_keys(sorted_benchmarks, controlled_variables)
181     for i, x in enumerate(benchmarks):
182         benchmarks[i] = group_by_keys(x, [group])
183         for j, y in enumerate(benchmarks[i]):
184             benchmarks[i][j] = group_by_keys(y, [variable])
185
186     # 2. statistically combine multiple measurements
187     # for the exact same benchmark and parameters,
188     # and store information about the roles of keys
189
190     for i, compatibles in enumerate(benchmarks):
191         for j, plotgroups in enumerate(compatibles):
192             for k, measured_values in enumerate(plotgroups):
193
194                 plotgroups[k] = aggregate_measurements(
195                     measured_values, measure, stat_fun=min)
196
197                 compatibles[j] = odict(
198                     (benchmark[variable], {
199                         'fixed': dict(
200                             (key, benchmark[key]) for key in controlled_variables),
201                         'info': dict((key, benchmark[key]) for key in info),
202                         'variable': variable,
203                         'measure': measure,
204                         'group': group,
205                         'variable': benchmark[variable],

```

```

206         measure: benchmark[measure],
207         group: benchmark[group]
208     }) for benchmark in plotgroups)
209
210     benchmarks[i] = odict(
211         sorted(((bms.values())[0][group], bms)
212             for bms in benchmarks[i]),
213         key=lambda x: x[0]))
214
215     return [x for x in benchmarks
216         if len((x.values())[0]) >= min_series_length]
217
218
219 def group_by_keys(sorted_benchmarks, keyset):
220     return [
221         list(y) for x, y in groupby(
222             sorted_benchmarks,
223             key=lambda b: [b[k] for k in keyset])]
224
225
226 def aggregate_measurements(benchmarks, measure, stat_fun=min):
227     values = []
228     benchmark = None
229     for benchmark in benchmarks:
230         values.append(benchmark[measure])
231
232     benchmark[measure] = stat_fun(values)
233
234     if len(values) != benchmark['multiplier']:
235         print ("Error: expecting", benchmark['multiplier'],
236             "measurements, got", len(values))
237         debugdata.write(pp.pformat(list(benchmarks)))
238         exit(1)
239
240     return benchmark
241
242
243 def comp_function(keys, left, right):
244     for key in keys:
245         if key not in left and key not in right:
246             continue
247         l, r = left[key], right[key]
248         if l < r:
249             return -1
250         if l > r:
251             return 1
252     return 0
253
254
255 def without(keys, d):
256     if keys == None:
257         return d
258     return dict(((key, val) for key, val in d.iteritems() if key not in keys))
259
260
261 def plot(
262     benchmarks, gnuplot_script, plotpath, metadata_file,
263     keys_to_remove=None, select_predicate=None,
264     group=None, variable=None, measure=None,
265     title=None, style=None, min_series_width=1,

```

```

266         key_placement='inside top left',
267         identifier=None,
268         revision=None, checksum=None, output='pdf'):
269
270     if len(benchmarks) > 0 and benchmarks[0].get('is_allocating'):
271         identifier += '-alloc'
272     if len(benchmarks) > 0:
273         reps = benchmarks[0].get('repetitions')
274
275     filtered_benchmarks = [
276         without(keys_to_remove, x)
277         for x in benchmarks
278         if select_predicate(x)]
279
280     variables = set([benchmark[variable] for benchmark in filtered_benchmarks])
281
282     if len(variables) < 2:
283         print 'Skipping plot without enough data variables', title
284         return
285
286     if len(filtered_benchmarks) == 0:
287         print 'Error, no benchmarks for', title
288         exit(1)
289
290     print 'Plotting', title
291
292     specs = {
293         'group': group,
294         'variable': variable,
295         'measure': measure}
296
297     data = extract_data(filtered_benchmarks, **specs)
298
299     index = -1
300
301     data_len = len([s for s in data if len(s.keys()) >= min_series_width])
302     for series in data:
303         if len(series.keys()) < min_series_width:
304             # there are not enough groups to display
305             continue
306         index += 1
307
308         plot.page += 1
309         axes_label = plot_axes.get(variable, '<unknown variable>')
310
311         headers, rows = make_table(
312             series, group, variable, measure, axes_label)
313
314         assert identifier is not None
315         id_suffix = ""
316         if data_len > 1:
317             id_suffix = "-{}".format(index)
318
319         gnuplot.output_plot(
320             headers, rows, plotpath, gnuplot_script,
321             title, specs, style, plot.page, identifier + id_suffix, axes_label,
322             output=output, key_placement=key_placement, reps=reps
323         )
324
325     metadata_file.write("\n\n{0}\n{1}\n\n".format(

```

```

326         title, identifier + id_suffix))
327
328     keyvalpairs = series.values()[0].values()[0]['fixed'].items() + [
329         ('variable', axes_label),
330         ('measure', measure),
331         ('grouping', group)]
332
333     for k, v in keyvalpairs:
334         if v != None:
335             metadata_file.write("{k:<25} {v}\n".format(k=k, v=v))
336
337     metadata_file.write(
338         "\n" + textualtable.make_textual_table(headers, rows))
339
340     id_headers, id_rows = make_table(
341         series, group, variable, 'class', axes_label)
342
343     def make_id(variable_value, item, variable):
344         ret = "/" .join([revision, item or '-'])
345         if variable == 'dynamic_size':
346             ret += "/" + str(variable_value)
347         return ret
348
349     id_rows = [
350         [row[0]] +
351         [make_id(row[0], item, variable) for item in row[1:]]
352         for row in id_rows]
353
354     ttable = textualtable.make_textual_table(id_headers, id_rows)
355     metadata_file.write("\n" + ttable)
356
357     if variable != 'direction' and variable != 'id':
358         x, polys, residuals = linear_fit(rows)
359
360         fitted_curves = []
361         for i, xval in enumerate(x):
362             current = [xval]
363             current.extend(rows[i][1:])
364             current.extend([numpy.polyval(polys[j], xval)
365                             for j in range(0, len(rows[i]) - 1)])
366             fitted_curves.append(current)
367
368         plot.page += 1
369         gnuplot.output_plot(
370             headers + headers[1:], fitted_curves, plotpath, gnuplot_script,
371             title, specs, 'fitted_lines', plot.page, identifier +
372             id_suffix + '-fit', axes_label, output=output, reps=reps)
373
374         def simplified_function(poly):
375             return "{:.3g} * x {:.3g}".format(poly[0], poly[1])
376         metadata_file.write(
377             "\nnpolynomial:\n" + textualtable.make_vertical_textual_table(
378                 headers[1:], [map(simplified_function, polys)]))
379         metadata_file.write(
380             "\nresiduals:\n" + textualtable.make_vertical_textual_table(
381                 headers[1:], [residuals]))
382         metadata_file.write(
383             "\nslope:\n" + textualtable.make_vertical_textual_table(
384                 headers[1:], [map(lambda p: p[0], polys)]))
385         metadata_file.write(

```

```

386         "\nintercept:\n" + textualltable.make_vertical_textual_table(
387             headers[1:], [map(lambda p: p[1], polys)])
388     return data
389
390 plot.page = 0
391
392 def convert_to_seconds(value):
393     if type(value) == int:
394         strval = str(value)
395         if convert_to_seconds == False:
396             return strval
397         strval = strval.zfill(10)
398         strlen = len(strval)
399         return float("{}.{ {}".format(
400             strval[0:strlen-9],
401             strval[strlen-9:]))
402     return value
403
404 def make_table(series, group, variable, measure, axes_label):
405     all_benchmark_variables_set = set()
406     for bm_list in series.itervalues():
407         all_benchmark_variables_set.update(bm_list.keys())
408
409     all_benchmark_variables = sorted(list(all_benchmark_variables_set))
410
411     rows = []
412
413     headers = (
414         [axes_label] +
415         [k for k in series.iterkeys()]
416     )
417
418     for v in all_benchmark_variables:
419         row = []
420         row.append(v)
421         for key, grp in series.iteritems():
422             val = grp.get(v, {}).get(measure, None)
423             if val is None:
424                 val = grp.get(v, {}).get('info', {}).get(measure, None)
425             if measure == 'response_time':
426                 val = convert_to_seconds(val)
427             row.append(val)
428         rows.append(row)
429
430     if variable == 'id':
431         rows = sorted(rows, key=lambda x: x[1] or -1)
432
433     return headers, rows
434
435
436 def binned_value(minimum, width, value):
437     return width * (int(value - minimum) / int(width)) + minimum
438
439
440 def plot_distributions(
441     all_benchmarks, output, plotpath, gnuplotcommands, bid,
442     metadata_file, plot_type=None, latex=None, **kwargs):
443
444     output_type = 'screen'
445     if plot_type != 'animate':

```

```

446         output_type = 'pdf'
447
448     gnuplot.init(gnuplotcommands, output, bid, output_type=output_type)
449     measure = 'response_time'
450
451     keyset = set(all_benchmarks[0].keys()) - \
452         set([measure, 'lineno', 'start', 'end'])
453     comparison_function = functools.partial(comp_function, keyset)
454     sorted_benchmarks = sorted(all_benchmarks, cmp=comparison_function)
455
456     for group in group_by_keys(sorted_benchmarks, keyset):
457         if plot_type != None:
458             keyf = lambda x: x['lineno']
459         else:
460             keyf = lambda x: x[measure]
461
462         frame_count = 1
463         if plot_type != None:
464             frame_count = 256
465
466         current_frame = frame_count
467         all_values = [b[measure] for b in sorted(group, key=keyf)]
468         while current_frame > 0:
469
470             if current_frame == frame_count:
471                 frame_ratio = 1
472             else:
473                 frame_ratio = float(current_frame) / frame_count
474             values = array(all_values[0:int(frame_ratio * len(all_values))])
475
476             bin_width = 500
477             min_x = numpy.amin(all_values)
478             max_x = numpy.amax(all_values)
479
480             bin_no = (max_x - min_x) / bin_width
481
482             hgram, bin_edges = numpy.histogram(values, bins=max(bin_no, 10))
483
484             mode = bin_edges[numpy.argmax(hgram)]
485             min_x = mode - 100000
486             max_x = mode + 100000
487
488             if current_frame == frame_count:
489                 metadata_file.write(
490                     'Direction {0}\n'.format(group[0]['direction']))
491
492                 gnuplotcommands.write(
493                     gnuplot.templates['binned_init'].format(
494                         title='%s %s' % (group[0]['id'], group[
495                             0]['direction']),
496                         binwidth=bin_edges[1] - bin_edges[0],
497                         min_x=min_x, max_x=max_x,
498                         max_y=numpy.max(hgram)))
499
500             if plot_type == 'animate':
501                 gnuplotcommands.write('pause -1\n')
502
503             elif plot_type == 'gradient':
504                 gnuplotcommands.write("set multiplot\n")
505

```

```

506         current_frame -= 1
507
508         if plot_type == None:
509             gnuplotcommands.write(
510                 gnuplot.templates['binned_frame'].format(
511                     datapoints='', color='#000033',
512                     values='\n'.join(['{} {} {}'.format(val, count, val)
513                                     for val, count in zip(
514                                         bin_edges, hgram)])))
515
516         elif plot_type == 'gradient':
517             gnuplotcommands.write(
518                 gnuplot.templates['binned_frame'].format(
519                     datapoints='',
520                     color=gnuplot.hex_color_gradient(
521                         (125, 0, 0), (255, 255, 0), 1 - frame_ratio),
522                     values='\n'.join(['{} {} {}'.format(val, count, val)
523                                     for val, count in zip(
524                                         bin_edges, hgram)])))
525
526         gnuplotcommands.write("set xtics\n")
527         gnuplotcommands.write("set ytics\n")
528
529
530 def plot_benchmarks(
531     all_benchmarks, output, plotpath, gnuplotcommands, bid, metadata_file,
532     plot_type=None, revision=None, checksum=None, latex=None):
533
534     output_type = 'pdf'
535     if latex == 'plotlatex':
536         output_type = 'latex'
537     elif latex == 'plotsvg':
538         output_type = 'svg'
539
540     gnuplot.init(gnuplotcommands, output, bid, output_type=output_type)
541
542     type_counts = ["parameter_type_{t}_count".format(t=tp) for tp in types]
543     keys_to_remove = type_counts[:]
544     keys_to_remove.extend(
545         ['parameter_type_count', 'single_type', 'dynamic_variation'])
546
547     benchmarks = [bm for bm in all_benchmarks if bm['no'] != -1]
548     defaults = [benchmarks, gnuplotcommands, plotpath]
549
550     overhead_estimates = {}
551     overhead_benchmarks = [
552         bm for bm in all_benchmarks
553         if bm['no'] == -1 and 'Overhead' in bm['id']]
554     for loop_type in ['AllocOverhead', 'NormalOverhead']:
555         for from_lang in ['C', 'J']:
556             language_name = from_lang
557             if language_name == 'J': language_name = 'Java'
558             overhead_estimates[from_lang] = {}
559             overhead_data = plot(
560                 overhead_benchmarks, gnuplotcommands, plotpath, metadata_file,
561                 style='simple_groups',
562                 key_placement=None,
563                 title='Mittauksen perusrasite ({}).format(language_name),
564                 identifier='{}-{}'.format(loop_type.lower(), from_lang.lower()),
565                 keys_to_remove=[],

```

```

566         select_predicate=(
567             lambda x: x['from'] == from_lang and loop_type in x['id']),
568         group='from',
569         measure='response_time',
570         variable='description',
571         revision=revision,
572         checksum=checksum,
573         output=output_type)
574
575     if overhead_data == None:
576         continue
577     if len(overhead_data) > 1:
578         print ('Error, more loop types than expected.',
579             len(overhead_data))
580         exit(1)
581
582     series = overhead_data[0]
583     headers, rows = make_table(series,
584                               'from',
585                               'description',
586                               'response_time',
587                               'workload')
588     est = estimate_measuring_overhead(rows[1:])
589     overhead_estimates[from_lang][loop_type] = est[0]
590     metadata_file.write('Overhead ' + from_lang + ' ' + str(est[0]))
591
592 for i, ptype in enumerate(types):
593     plot(
594         benchmarks, gnuplotcommands, plotpath, metadata_file,
595         title='{0}-tyyppiset kutsuparametrit'.format(ptype),
596         identifier='basic-call-{0}'.format(ptype),
597         style='simple_groups',
598         keys_to_remove=(
599             keys_to_remove +
600             ['dynamic_size'] +
601             ['has_reference_types']),
602         select_predicate=lambda x: (
603             x['single_type'] in [ptype, 'any'] and
604             x['dynamic_size'] == 0),
605         group='direction',
606         variable='parameter_count',
607         measure='response_time',
608         revision=revision, checksum=checksum, output=output_type)
609
610 for fr, to in DIRECTIONS:
611     direction = format_direction(fr, to, latex)
612     plot(
613         benchmarks, gnuplotcommands, plotpath, metadata_file,
614         title='Vaihteleva argumentin koko kutsusuunnassa ' + direction,
615         identifier='variable-argument-size-{0}-{1}'.format(fr.lower(),
616                                                         to.lower()),
617         style='simple_groups',
618         keys_to_remove=type_counts,
619         select_predicate=(
620             lambda x: (
621                 x['direction'] == direction and
622                 x['has_reference_types'] == 1 and
623                 x['single_type'] in reference_types and
624                 x['parameter_count'] == 1)),
625         group='single_type',

```



```

626         variable='dynamic_size',
627         measure='response_time',
628         revision=revision, checksum=checksum, output=output_type)
629
630     for fr, to in DIRECTIONS:
631         direction = format_direction(fr, to, latex)
632         plot(
633             benchmarks, gnuplotcommands, plotpath, metadata_file,
634             title='Vaihteleva paluuarvon koko kutsusuunnassa ' + direction,
635             identifier='variable-return-value-size-{}-{}'.format(fr.lower(),
636                                                                     to.lower()),
637             style='simple_groups',
638             keys_to_remove=type_counts,
639             select_predicate=(
640                 lambda x: x['has_reference_types'] == 1
641                 and x['direction'] == direction
642                 and x['return_type'] != 'void'),
643             group='return_type',
644             variable='dynamic_size',
645             measure='response_time',
646             revision=revision, checksum=checksum, output=output_type)
647
648     keys_to_remove = type_counts[:]
649     keys_to_remove.append('has_reference_types')
650     keys_to_remove.append('dynamic_variation')
651
652     for fr, to in DIRECTIONS:
653         direction = format_direction(fr, to, latex)
654         plot(
655             benchmarks, gnuplotcommands, plotpath, metadata_file,
656             style='simple_groups',
657             title='Parametrityyppien vertailu ' + direction,
658             identifier='basic-call-all-types-{}-{}'.format(fr.lower(),
659                                                                     to.lower()),
660             keys_to_remove=keys_to_remove,
661             select_predicate=(
662                 lambda x: x['direction'] == direction),
663             group='single_type',
664             variable='parameter_count',
665             measure='response_time',
666             revision=revision, checksum=checksum, output=output_type)
667
668     plot(
669         benchmarks, gnuplotcommands, plotpath, metadata_file,
670         style='named_columns',
671         title='Paluuarvon tyytit',
672         identifier='return-value-types',
673         keys_to_remove=['has_reference_types', 'dynamic_variation'],
674         select_predicate=(
675             lambda x: x['dynamic_size'] == 0 and
676             x['return_type'] != 'void'),
677         group='return_type',
678         measure='response_time',
679         variable='direction',
680         min_series_width=2,
681         revision=revision, checksum=checksum, output=output_type)
682
683     def utf(b):
684         return 'UTF' in b['id'] or 'Utf' in b['id']
685

```

```

686 filters = {
687     'utf': utf,
688     'arrayregion': lambda x: 'ArrayRegion' in x['id'],
689     'bytebufferview': lambda x: 'ByteBufferView' in x['id'],
690     'unicode': lambda b: not utf(b) and 'String' in b['id'],
691     'arrayelements': (lambda x:
692         'ArrayElements' in x['id'] or
693         'ArrayLength' in x['id'] or
694         'ReadPrimitive' in x['id']),
695 }
696 def uncategorized(x):
697     for f in filters.values():
698         if f(x):
699             return False
700     return True
701
702 benchmarks = {}
703 for key, f in filters.iteritems():
704     benchmarks[key] = [
705         bm for bm in all_benchmarks
706         if bm['no'] == -1 and f(bm)]
707
708 benchmarks['uncategorized'] = [
709     bm for bm in all_benchmarks
710     if bm['no'] == -1 and 'Overhead' not in bm['id'] and uncategorized(bm)]
711
712 custom_benchmarks = benchmarks['uncategorized']
713
714 for fr, to in DIRECTIONS:
715     direction = format_direction(fr, to, latex)
716     plot(
717         custom_benchmarks, gnuplotcommands, plotpath, metadata_file,
718         style='simple_groups',
719         title='Erityiskutsut suunnassa ' + direction,
720         identifier='special-calls-{}-{}'.format(fr.lower(), to.lower()),
721         select_predicate=(
722             lambda x: (x['direction'] == direction and
723                 x['dynamic_variation'] == 1)),
724         group='id',
725         measure='response_time',
726         variable='dynamic_size',
727         revision=revision, checksum=checksum, output=output_type)
728
729     plot(
730         benchmarks['arrayregion'], gnuplotcommands, plotpath,
731         metadata_file,
732         style='simple_groups',
733         title='Erityiskutsut suunnassa ' + direction,
734         identifier='special-calls-arrayregion-{}-{}'.format(fr.lower(),
735             to.lower()),
736         select_predicate=(
737             lambda x: (x['direction'] == direction and
738                 x['dynamic_variation'] == 1)),
739         group='id',
740         measure='response_time',
741         variable='dynamic_size',
742         revision=revision, checksum=checksum, output=output_type)
743
744     plot(
745         benchmarks['arrayelements'], gnuplotcommands, plotpath,

```

```

746     metadata_file,
747     style='simple_groups',
748     title='Erityiskutsut suunnassa ' + direction,
749     identifier='special-calls-arrayelements-{}-{}'.format(fr.lower(),
750                                                         to.lower()),
751     select_predicate=(
752         lambda x: (x['direction'] == direction and
753                   x['dynamic_variation'] == 1)),
754     group='id',
755     measure='response_time',
756     variable='dynamic_size',
757     revision=revision, checksum=checksum, output=output_type)
758
759 plot(
760     benchmarks['utf'], gnuplotcommands, plotpath, metadata_file,
761     style='simple_groups',
762     title='UTF-merkkijonot suunnassa ' + direction,
763     identifier='special-calls-utf-{}-{}'.format(fr.lower(),
764                                                 to.lower()),
765     select_predicate=(
766         lambda x: (x['direction'] == direction and
767                   x['dynamic_variation'] == 1)),
768     group='id',
769     measure='response_time',
770     variable='dynamic_size',
771     revision=revision, checksum=checksum, output=output_type)
772
773 plot(
774     benchmarks['unicode'], gnuplotcommands, plotpath, metadata_file,
775     style='simple_groups',
776     key_placement='inside bottom left',
777     title='Unicode-merkkijonot suunnassa ' + direction,
778     identifier='special-calls-unicode-{}-{}'.format(fr.lower(),
779                                                     to.lower()),
780     select_predicate=(
781         lambda x: (x['direction'] == direction and
782                   x['dynamic_variation'] == 1)),
783     group='id',
784     measure='response_time',
785     variable='dynamic_size',
786     revision=revision, checksum=checksum, output=output_type)
787
788 plot(
789     benchmarks['bytebufferview'], gnuplotcommands, plotpath,
790     metadata_file,
791     style='simple_groups',
792     title='Erityiskutsut suunnassa ' + direction,
793     identifier='special-calls-bytebufferview-{}-{}'.format(fr.lower(),
794                                                           to.lower()),
795     select_predicate=(
796         lambda x: (x['direction'] == direction and
797                   x['dynamic_variation'] == 1 and
798                   'Bulk' not in x['id'])),
799     group='id',
800     measure='response_time',
801     variable='dynamic_size',
802     revision=revision, checksum=checksum, output=output_type)
803
804 plot(
805     benchmarks['bytebufferview'], gnuplotcommands, plotpath,

```

```

806     metadata_file,
807     style='simple_groups',
808     title='Erityiskutsut suunnassa ' + direction,
809     identifier='special-calls-bulk-bytebufferview-{}-{}'.format(
810         fr.lower(), to.lower()),
811     select_predicate=(
812         lambda x: (x['direction'] == direction and
813                    x['dynamic_variation'] == 1 and
814                    'Bulk' in x['id'])),
815     group='id',
816     measure='response_time',
817     variable='dynamic_size',
818     revision=revision, checksum=checksum, output=output_type)
819
820 plot(
821     custom_benchmarks, gnuplotcommands, plotpath, metadata_file,
822     style='histogram',
823     title='Erityiskutsujen vertailu eri kutsusuunnissa',
824     identifier='special-calls-non-dynamic',
825     select_predicate=(
826         lambda x: (
827             x['dynamic_variation'] == 0 and
828             'Field' in x['id'])),
829     group='direction',
830     measure='response_time',
831     variable='id',
832     revision=revision, checksum=checksum, output=output_type)
833
834
835 MEASUREMENT_FILE = 'measurements.txt'
836 DEVICE_PATH = '/sdcard/results'
837 PLOTPATH = '/tmp'
838 TOOL_NAMESPACE = 'fi.helsinki.cs.tituomin.nativebenchmark.measuringtool'
839
840
841 def sync_measurements(dev_path, host_path, filename, update=True):
842     old_path = host_path + '/' + filename
843     tmp_path = '/tmp/' + filename
844     if not update and os.path.exists(old_path):
845         print 'No sync necessary'
846         return
847
848     kwargs = {}
849     if FNULL is not None:
850         kwargs['stdout'] = FNULL
851         kwargs['stderr'] = FNULL
852
853     try:
854         success = call(['adb', 'pull',
855                        dev_path + '/' + filename,
856                        tmp_path], **kwargs)
857     except OSError:
858         success = -1
859     if success == 0:
860         if os.path.exists(old_path):
861             size_new = os.path.getsize(tmp_path)
862             size_old = os.path.getsize(old_path)
863             if size_new < size_old:
864                 print ("Warning: new file contains less data than "
865                       "the old. Aborting.")

```

```

866         exit(2)
867         shutil.move(tmp_path, old_path)
868
869     else:
870         print "Could not get new measurements, continuing with old."
871
872 def render_perf_reports_for_measurement(identifier, measurements,
873                                       measurement_path, output_path,
874                                       output_command=False):
875     path = identifier.split("/")
876     if len(path) < 2:
877         print 'Invalid identifier {}'.format(identifier)
878         exit(1)
879     if len(path) == 3:
880         revision, class_, dynamic_size = path
881     elif len(path) == 2:
882         revision, class_ = path
883         dynamic_size = None
884
885     def match_measurement(measurement):
886         m = measurement[0]
887         return (m.get('code-revision') == revision and
888                 m.get('tool') == 'LinuxPerfRecordTool')
889
890     def match_measurement_run(m):
891         if m.get('class').lower() != class_.lower():
892             return False
893         if dynamic_size and m.get('dynamic_size') != int(dynamic_size):
894             return False
895         if 'Filename' not in m or m['Filename'] is None:
896             return False
897         return True
898
899     datafiles = []
900     for measurement in filter(match_measurement, measurements):
901         mid = measurement[0].get('id')
902         zpath = os.path.join(measurement_path, 'perfddata-{}.zip'.format(mid))
903         try:
904             measurement_zipfile = zipfile.ZipFile(zpath, 'r')
905             datafiles.append({
906                 'zip': measurement_zipfile,
907                 'zip_path': zpath,
908                 'mid': mid,
909                 'csv': measurement_zipfile.open(
910                     '{}/benchmarks-{}.csv'.format(mid))
911             })
912         except zipfile.BadZipfile:
913             print 'Bad zip file %s' % zpath
914         except IOError as e:
915             print 'Problem with zip file %s' % zpath
916             print e
917
918     benchmarks = []
919     for df in datafiles:
920         benchmarks.append({
921             'zip': df['zip'],
922             'mid': df['mid'],
923             'metadata': read_datafiles([df['csv']], silent=output_command)
924         })
925

```

```

926 matching_benchmarks = []
927 for bm in benchmarks:
928     for row in bm['metadata']:
929         if match_measurement_run(row):
930             matching_benchmarks.append({
931                 'zip': bm['zip'],
932                 'mid': bm['mid'],
933                 'filename': row['Filename']
934             })
935
936 for record in matching_benchmarks:
937     perf_file = record['zip'].extract('{}/{}/{}'.format(record['mid'],
938                                                         record['filename']),
939                                     '/tmp')
940
941     try:
942         command_parts = [
943             "perf report",
944             "-i {}",
945             "--header",
946             "--syms=/home/tituomin/droid-symbols",
947             "--kallsyms=/home/tituomin/droid/linux-kernel/kallsyms"
948         ]
949         command_parts.extend([
950             "-g graph,0,caller",
951             "--stdio",
952             "| c++filt",
953             ">/tmp/out.txt"
954         ])
955         command = " ".join(command_parts).format(perf_file)
956         if output_command:
957             print command
958             exit(0)
959         else:
960             call([command], shell=True)
961     except OSError as e:
962         print e.filename, e.message, e.args
963
964 for f in datafiles:
965     f['zip'].close()
966     print "Profile for identifier", identifier
967     with open('/tmp/out.txt', 'r') as f:
968         print f.read()
969     exit(0)
970
971 if __name__ == '__main__':
972     if len(argv) < 4 or len(argv) > 6:
973         print argv[0]
974         print ("\n Usage: %s input_path output_path "
975               "limit [pdfviewer] [separate]\n").format(argv[0])
976         exit(1)
977
978 FNULL = open(os.devnull, 'w')
979
980 method = argv[0]
981 measurement_path = os.path.normpath(argv[1])
982 output_path = argv[2]
983
984 if 'plotlatex' in method:
985     latex = 'plotlatex'
986     method = 'curves'

```

```

986     elif 'plotsvg' in method:
987         latex = 'plotsvg'
988         method = 'curves'
989     else:
990         latex = None
991
992     output_command = False
993     if len(argv) > 5:
994         if argv[5] == 'show-command':
995             output_command = True
996
997     limit = argv[3]
998     if len(argv) > 4:
999         pdfviewer = argv[4]
1000    else:
1001        pdfviewer = None
1002
1003    if len(argv) == 6:
1004        group = (not argv[5] == "separate")
1005    else:
1006        group = True
1007
1008    if output_command:
1009        system_stdout = sys.stdout
1010        system_stderr = sys.stderr
1011        sys.stdout = FNULL
1012        sys.stderr = FNULL
1013
1014    sync_measurements(DEVICE_PATH, measurement_path, MEASUREMENT_FILE)
1015
1016    f = open(os.path.join(measurement_path, MEASUREMENT_FILE))
1017
1018    try:
1019        measurements = read_measurement_metadata(f, group)
1020    finally:
1021        f.close()
1022
1023    limited_measurements = (
1024        filter(lambda x: int(x[0].get('repetitions', 0)) >= int(limit),
1025             measurements.values()))
1026
1027    # ID = revision/checksum/class[/dynamic_size]
1028    if 'perf_select' in method:
1029        identifier = argv[4]
1030        if output_command:
1031            sys.stdout = system_stdout
1032            sys.stderr = system_stderr
1033            FNULL.close()
1034        render_perf_reports_for_measurement(
1035            identifier, limited_measurements, measurement_path,
1036            output_path, output_command=output_command)
1037        exit(0)
1038
1039    csv_files = set()
1040    for f in glob.iglob(measurement_path + '/benchmarks-*.csv'):
1041        try:
1042            csv_files.add(f.split('.')[0].split('benchmarks-')[1])
1043        except IndexError:
1044            pass
1045

```

```

1046 if len(limited_measurements) > 20:
1047     i = len(limited_measurements) - 20 + 1
1048     splice = limited_measurements[-20:]
1049 else:
1050     i = 1
1051     splice = limited_measurements
1052
1053 print "\nAvailable compatible measurements. Choose one"
1054 for m in splice:
1055     b = m[0]
1056     warning = ""
1057     if int(b.get('rounds')) == 0:
1058         warning = " <---- WARNING INCOMPLETE MEASUREMENT"
1059     print ""
1060     [{idx}]:      total measurements: {num}
1061                   local: {local}
1062                   repetitions: {reps}
1063                   description: {desc}
1064                   rounds: {rounds}{warning}
1065                   id: {mid}
1066                   checksum: {ck}
1067                   revision: {rev}
1068                   tool: {tool}
1069                   cpu: {freq} KHz
1070                   set: {bset}
1071                   filter: {sfilter}
1072                   dates: {first} -
1073                        {last}
1074     """.format(
1075         local=b.get('id') in csv_files,
1076         num=len(m),
1077         mid=b.get('id'),
1078         idx=i,
1079         warning=warning,
1080         last=m[-1]['end'],
1081         rounds=reduce(lambda x, y: y + x, [int(b['rounds']) for b in m]),
1082         reps=b.get('repetitions'),
1083         ck=b.get('code-checksum'),
1084         rev=b.get('code-revision'),
1085         tool=b.get('tool'),
1086         freq=b.get('cpu-freq'),
1087         bset=b.get('benchmark-set'),
1088         desc=b.get('description'),
1089         sfilter=b.get('substring-filter'),
1090         first=b.get('start')
1091     )
1092
1093     i += 1
1094
1095 try:
1096     response = raw_input("Choose set 1-{last} >> ".format(last=i - 1))
1097 except EOFError:
1098     print 'Exiting.'
1099     exit(1)
1100
1101 benchmark_group = limited_measurements[int(response) - 1]
1102
1103 filenames = []
1104 ids = []
1105 multiplier = 0

```



```

1106 for measurement in benchmark_group:
1107     if 'LinuxPerfRecordTool' in measurement['tool']:
1108         basename = "perfdata-{n}.zip"
1109     else:
1110         basename = "benchmarks-{n}.csv"
1111     filenames.append(
1112         basename.format(n=measurement['id']))
1113     if 'logfile' in measurement:
1114         filenames.append(measurement['logfile'])
1115     ids.append(measurement['id'])
1116     multiplier += int(measurement['rounds'])
1117
1118 files = []
1119 for filename in filenames:
1120     sync_measurements(DEVICE_PATH, measurement_path,
1121                       filename, update=False)
1122     if filename not in [m.get('logfile') for m in benchmark_group]:
1123         files.append(open(os.path.join(measurement_path, filename)))
1124
1125 first_measurement = benchmark_group[0]
1126
1127 global_values = {
1128     'repetitions': first_measurement['repetitions'],
1129     'is_allocating': first_measurement['benchmark-set'] == 'ALLOC',
1130     'multiplier': multiplier
1131 }
1132
1133 perf = False
1134 if 'LinuxPerfRecordTool' in first_measurement['tool']:
1135     print 'Perf data downloaded.'
1136     perf = True
1137 if not perf:
1138     try:
1139         benchmarks = read_datafiles(files)
1140
1141     finally:
1142         for f in files:
1143             f.close()
1144
1145 benchmark_group_id = os.getenv('PLOT_ID', str(uuid.uuid4()))
1146 plot_prefix = 'plot-{0}'.format(benchmark_group_id)
1147
1148 if latex is not None:
1149     output_filename = os.path.join(output_path, plot_prefix)
1150 else:
1151     output_filename = os.path.join(output_path, plot_prefix + '.pdf')
1152 plot_filename = plot_prefix + '.gp'
1153
1154 plotfile = open(os.path.join(output_path, plot_filename), 'w')
1155 metadata_file = open(os.path.join(
1156     output_path, plot_prefix + '-metadata.txt'), 'w')
1157
1158 measurement_ids = " ".join(ids)
1159 metadata_file.write("-- mode: perf-report; --\n\n")
1160 metadata_file.write("id: {0}\n".format(benchmark_group_id))
1161 metadata_file.write("measurements: {0}\n".format(measurement_ids))
1162
1163 benchmarks = preprocess_benchmarks(benchmarks, global_values,
1164                                     latex=latex)
1165

```

```

1166         animate = False
1167         if pdfviewer == 'anim':
1168             plot_type = 'animate'
1169             pdfviewer = None
1170         elif pdfviewer == 'gradient':
1171             plot_type = 'gradient'
1172             pdfviewer = None
1173         else:
1174             plot_type = None
1175
1176         if 'curves' in method:
1177             function = plot_benchmarks
1178         elif 'distributions' in method:
1179             function = plot_distributions
1180         if perf or not function:
1181             exit(0)
1182
1183         function(
1184             benchmarks,
1185             output_filename,
1186             PLOTPATH,
1187             plotfile,
1188             benchmark_group_id,
1189             metadata_file,
1190             plot_type=plot_type,
1191             revision=first_measurement['code-revision'],
1192             checksum=first_measurement['code-checksum'],
1193             latex=latex)
1194
1195         plotfile.flush()
1196         plotfile.close()
1197         if plot_type == 'animate':
1198             print "Press enter to start animation."
1199         call(["gnuplot", plotfile.name])
1200         if pdfviewer:
1201             call([pdfviewer, str(output_filename)])
1202         print "Final plot",
1203         if 'animate' != plot_type:
1204             print str(output_filename)
1205         else:
1206             print str(plot_filename)
1207         print(benchmark_group_id)
1208         exit(0)

```

/textualtable.py

```

1 #/usr/bin/python
2
3 def make_textual_table(headers, rows):
4     result = ""
5     max_widths = []
6
7     for x in headers:
8         max_widths.append(len(str(x)))
9

```

```

10     for row in rows:
11         for i, x in enumerate(row):
12             l = len(str(x))
13             if max_widths[i] < l:
14                 max_widths[i] = l
15
16     row_format = ["{:>{w}}".format(w=w) for w in max_widths]
17     row_format = "".join(row_format) + "\n"
18
19     result += row_format.format(*headers)
20     for row in rows:
21         result += row_format.format(*row)
22     return result
23
24 def make_vertical_textual_table(headers, elements):
25     result = ""
26     max_width = max((len(x) for x in headers))
27
28     header_format = "{:>{w}}".format(w=max_width)
29
30     for i in range(0, len(headers)):
31         result += header_format.format(headers[i])
32         for group in elements:
33             result += "    "
34             result += str(group[i])
35             result += "\n"
36
37     return result

```